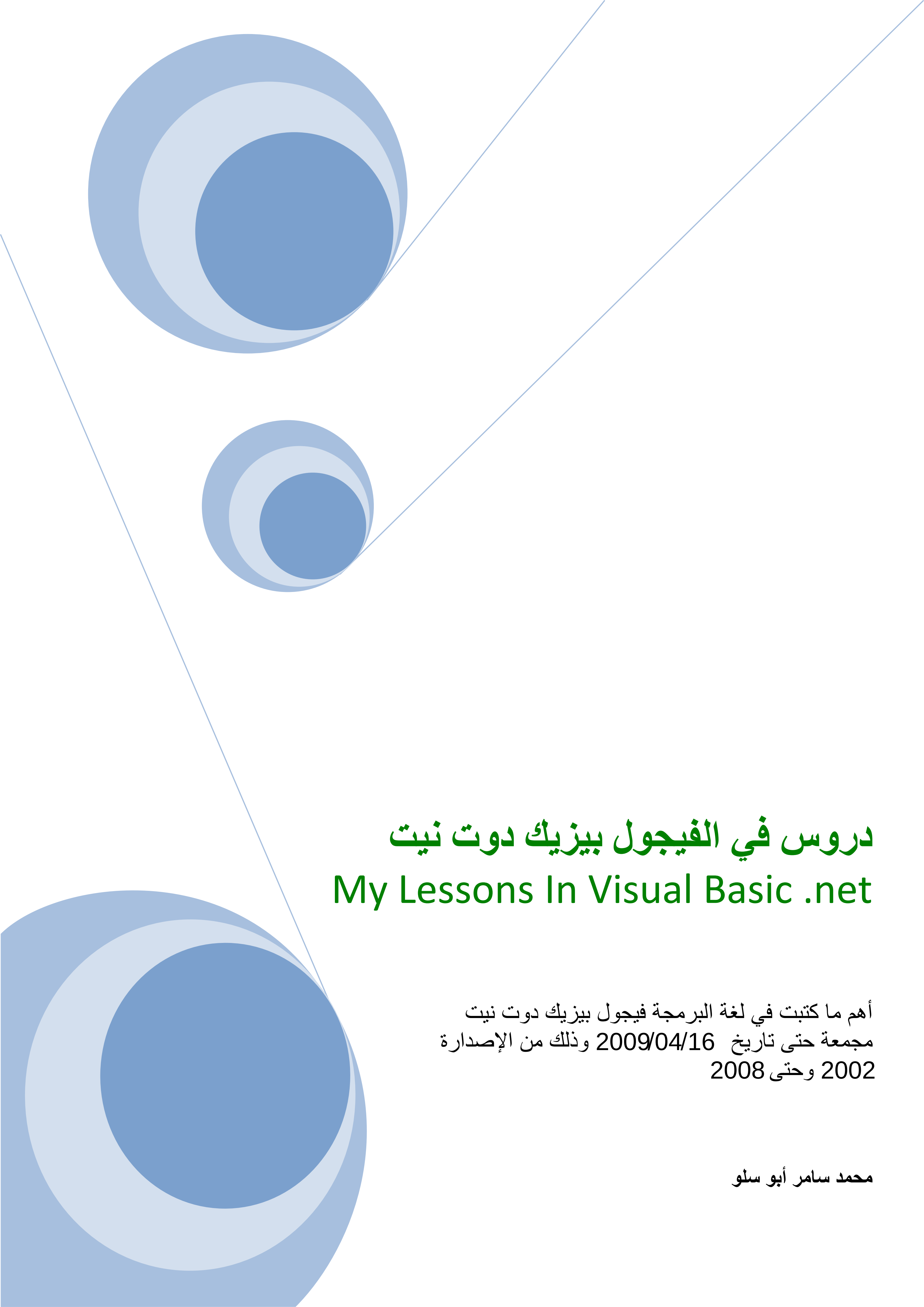


دروس في الفيجول بيزيك دوت نيت
My Lessons In Visual Basic .net

أهم ما كتبت في لغة البرمجة فيجول بيزيك دوت نيت
مجمعة حتى تاريخ 2009/04/16 وذلك من الإصدارة 2002 وحتى 2008

محمد سامر أبو سلو

دروس في الفيجول بيزيك دوت نيت

My Lessons in Visual Basic .net

بقلم

محمد سامر أبو سلو

[*Samer.selo@yahoo.com*](mailto:Samer.selo@yahoo.com)

[*Samerselo2005@hotmail.com*](mailto:Samerselo2005@hotmail.com)

الإهداء

إلى والدي ووالدتي الحبيبين وإخوتي

إلى جميع أصدقائي في جميع المنتديات

إلى كل من يتوق للاستفادة من هذا الكتاب

شكر خاص

إلى والدي ووالدتي

إلى الإخوة الذين اقترحوا تجميع المقالات في كتاب واحد

إلى كل من ساعدني في التجميع والفهرسة

مقدمة

هذا الكتاب هو تجميع لأهم مواضيعي ومقالاتي في منتديات البرمجة العربية بلغة Visual Basic .net بإصداراتها المختلفة ابتداء من 2002 وحتى 2008 وهو لا يشكل مرجعا متكاملا للغة ولكن المواضيع الموجودة فيه تهم كل مبرمج يتعامل مع هذه اللغة راجيا الفائدة لكل من يقرأه حيث جمعت المقالات في عدة أقسام متنوعة ليضم كل قسم مجموعة من المقالات المتشابهة بالموضوع لهذا تستطيع قراءة الأقسام بدءا من القسم أو الموضوع الذي يعجبك مع الانتباه إلى خصوصية قسم تقنية Linq حيث يتوجب عليك دراسة بعض المواضيع المنوه عنها في بداية القسم قبل البدء به ومتابعته بالتسلسل الوارد في الكتاب .

كما يفضل أن تعرف على الأقل بعض أساسيات اللغة كي تستطيع المتابعة والفهم فيجب أن تكون لديك خلفية جيدة عن أساسيات لغة الفيجول بايزيك من حلقات التكرار والحلقات الشرطية وتعريف المتغيرات ومجالها كما أن بعض المواضيع المتقدمة يلزمها أن تكون ملما ببرمجة قواعد البيانات وطريقة كتابة استعلامات Select وخاصة عندما تنتقل إلى القسم الخاص بـ Linq مع وجود مواضيع تحتاج إلى إلمام بطريقة تعريف وإنشاء الفئات Classes كما يفضل أن تكون لديك معرفة بمبادئ HTML و XML لتعلقها ببعض الدروس الموجودة في الكتاب

القسم الأول – معالجة الأخطاء

ويضم المواضيع التالية:

- معالجة الأخطاء
- تنقيح الأخطاء في برنامجك
- الاستثناءات Exceptions اصطياد الأخطاء ومعالجتها

معالجة الأخطاء

متى نستخدم معالجات الأخطاء

يمكنك استخدام معالجات الأخطاء في أي وضع يولد احتمال لحدوث خطأ يوقف تنفيذ البرنامج سواء كان ذلك الخطأ متوقعا أم غير متوقع. وبشكل عام تستخدم معالجات الأخطاء لإدارة أحداث خارجية قد تؤثر على مسار تنفيذ البرنامج كفشل في الوصول للشبكة أو قرص مضغوط غير موجود أو طابعة أو ماسح ضوئي غير مشغلين مثلا ومن هذه المشاكل المحتملة التي تحتاج إلى معالجات أخطاء:

- **شبكة/انترنت** مشكلة في المخدمات أو تجهيزات الاتصال
- **قواعد بيانات** عدم القدرة على إنشاء اتصال أو تنفيذ استعلام أو أن قاعدة البيانات تعيد خطأ ما
- **سواقات الأقراص** قرص غير مهيأ أو مهيأ بصورة غير صحيحة أو قرص قابل للإزالة غير مدخل بشكل جيد أو قطاعات تالفة أو حتى قرص ملئ
- **مشاكل المسارات** مسار خاطئ أو غير صحيح
- **مشاكل الطابعة** طابعة مطفأة أو بدون ورق أو غير متوفرة لسبب ما
- **مشاكل برمجيات** مكونات أو مكتبات يعتمد عليها ملف للتنفيذ ناقصة أو غير منصبة بصورة صحيحة أو
- وجود تعارض أو عدم توافقية بين بعض المكتبات
- **مشاكل أمان** البرنامج يحاول القيام بعملية غير مسموحة أو أن المستخدم الذي يشغل البرنامج لا يمتلك
- الصلاحيات الكافية لإتمام تنفيذ العملية
- **مشاكل ذاكرة** مصادر نظام غير كافية
- **مشاكل الحافظة** مشاكل في تبادل البيانات مع حافظة ويندوز
- **مشاكل منطقية** مشاكل صيغة أو مشاكل منطقية لم يستطع المترجم كشفها

كتلة Try ... Catch

كتلة الكود التي تستخدم لمعالجة أخطاء زمن التنفيذ تدعى Try ... Catch حيث يمكنك كتابة عبارة Try ضمن إجراء معالجة الحدث قبل الكود الذي تتوقع أن يولد مشكلة وتليها مباشرة عبارة Catch فإن حدث خطأ في زمن التنفيذ فيتم تنفيذ مجموعة من العبارات ضمن كتلة Catch ثم ستنفذ مجموعة من العبارات الاختيارية في كتلة Finally كما يمكن في بعض الحالات تعشيش عدة بلوكات Try ... Catch داخل بعضها و تكون الصيغة العامة لكتلة Try ... Catch

```
Try
        Statements that might produce a run-time error
        العبارات الممكن أن تولد خطأ
Catch
        Statements to run if a run-time error occurs
        العبارات التي تنفذ في حالة حدوث خطأ
Finally
        Optional statements to run whether an error occurs or not
        عبارات اختيارية ستنفذ إن حدث خطأ أم لا
End Try
```

حيث تشكل عبارة Try بداية تعريف معالجة الخطأ في حين أن العبارات Try و Catch و End Try هي عبارات إجبارية والعبارة Finally اختيارية. ويدعى الكود المتواجد بين عبارتي Try و Catch بالكود المحمي بسبب أن أخطاء زمن التنفيذ المتولدة ضمن ذلك الكود لا تتسبب في توقف البرنامج عن العمل

فمثلا إن حاولنا فتح ملف صورة وتحميله في صندوق الصورة يمكننا وضع ذلك الكود ضمن كتلة Try ... Catch وذلك لحماية البرنامج من الأخطاء التي قد تحدث في زمن التنفيذ

```
Try
        PictureBox1.Image = _
                System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch
```

```
	MsgBox("Please insert the disk in drive D")
End Try
```

والمثال التالي يبين لنا كيفية تعشيش كتل Try ... Catch

```
Try
	PictureBox1.Image = _
		System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch
	MsgBox("Please insert the disk in drive D, Then Click Ok")
	Try
		PictureBox1.Image = _
			System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
	Catch
		MsgBox("File Load feature disabled")
	End Try
End Try
```

الغرض Err

من الأغراض الموروثة المفيدة الباقية من نسخ فيجول بايزيك السابقة الغرض Err بنسخته المحدثة بمعلومات مفصلة لمعالجة الأخطاء لكل خطأ زمن تنفيذ يحدث في البرنامج ومع ذلك هناك طرق أحدث لإدارة الأخطاء في الفريموورك مثل الغرض Exception القوي وتشكل الخصائص Err.Number و Err.Description الخصائص الأكثر إفادة لتحديد أخطاء زمن التنفيذ فالخاصية Err.Number تحتوي على رقم الخطأ الخاص بآخر خطأ زمن تنفيذ حدث مؤخرا والخاصية Err.Description تحتوي على رسالة قصيرة تطابق رقم الخطأ الذي حدث مؤخرا وتلك الخاصيتان تمكناك من التعرف على الأخطاء التي حدثت مؤخرا وتحديد الاستجابة المناسبة لها وقد تعطي المستخدم رسالة عن كيف يمكن أن يتصرف في حالة حدوث خطأ معين كما يمكنك تفريغ الخطأ بواسطة الطريقة Err.Clear ولكن إن استخدمت الغرض Err داخل كتلة Catch فيصبح من غير الضروري تفريغ الخطأ لأنه لا يتم الدخول إلى كتلة Catch إلا إن حدث خطأ والمثال التالي يتعرف على عدة أخطاء زمن التنفيذ باستخدام الغرض Err

```
Try
	PictureBox1.Image = _
		System.Drawing.Bitmap.FromFile("D:\FileOpen.bmp")
Catch When Err.Number = 53   'If File Not Found Error
	MsgBox("Check pathname and disk drive")
Catch When Err.Number = 7   'If File Out Of Memory Error
	MsgBox("Is this really a bitmap", , Err.Description)
Catch
	MsgBox("Problem loading file", , Err.Description)
End Try
```

العبارة Exit Try

وهي تستخدم للخروج من كتلة Try ... Catch بشكل مشابه للعبارات Exit For و Exit Sub المألوفة حيث باستخدامها تخرج كليا من كتلة Try ولكن إن كان قسم Finally موجودا فسيتم تنفيذه ولكن عبارة Exit Try تجعلك تقفز فوق بقية عبارات Try ... Catch الباقية

```
Try
	If PictureBox1.Enabled = False Then Exit Try
	PictureBox1.Image = _
		System.Drawing.Bitmap.FromFile("D:\fileopen.bmp")
Catch
	Retries +=1
	If Retries <= 2 Then
		MsgBox("Please insert the disc in drive D")
```

```
        Else
                MsgBox("File Load feature disabled")
                Button1.Enabled = False
        End If
End Try
```

مقارنة معالجات الأخطاء مع التقنيات الدفاعية للبرمجة

استخدام معالجات الأخطاء ليست الطريقة الوحيدة لحماية برنامجك من حدوث أخطاء زمن التنفيذ فقطعة الكود التالية تستخدم الطريقة File.Exists في مجال الأسماء System.IO في مكتبة فئات الفريموورك للتأكد من أن الملف موجود فعلا قبل محاولة فتحه

```
If file.Exists("D:\fileopen.bmp") Then

        PictureBox1.Image = _

                System.Drawing.Bitmap.FromFile("D:\fileopen.bmp")

Else

        MsgBox("Cannot find fileopen.bmp on drive D.")

End If
```

في الكود السابق لا تعتبر عبارة If معالج خطأ حقيقي لأنها لا تستطيع منع خطأ زمن التنفيذ من إيقاف تنفيذ البرنامج وبدلا عن ذلك فطريقة التحقق هذه التي يستخدمها بعض المبرمجين تدعى بالبرمجة الدفاعية. فهي تستخدم وظيفة مفيدة في مكتبة فئات الفريموورك للتأكد من العملية التي ستجري على الملف قبل محاولة فتحه الفعلية. وفي هذه الحالة خاصة فالتأكد من وجود الملف باستخدام الطريقة File.Exists هي أسرع من انتظار فيجول بايزيك لإطلاق الاستثناء واستعادته من خطأ زمن التنفيذ باستخدام معالجات الأخطاء.
وهنا يظهر لدينا سؤال: متى يجب علينا استخدام طريقة البرمجة الدفاعية ومتى يجب علينا استخدام معالجات الأخطاء؟ ويكون الجواب هو أنه يجب عليك استخدام مزيج من الطريقتين في كودك حيث تكون طريقة البرمجة الدفاعية هي الأكثر فعالية لمعالجة المشاكل المحتملة. وكما ذكرنا سابقا فالطريقة File.Exists أسرع من بلوك Try … Catch لمعالجة الأخطاء لذا فمن المنطقي استخدام تقنية البرمجة الدفاعية من أجل الأخطاء التي تتوقع حدوثها بشكل متكرر واستخدام التراكيب الخاصة بمعالجة الأخطاء إذا كان لديك أكثر من شرط لفحصه وتريد تزويد مستخدم برنامجك بعدد من الخيارات كاستجابة لذلك الخطأ كما تمكنك من معالجة الأخطاء التي قد لا تتوقعها.

تنقيح الأخطاء في برنامجك، Debugging Your Application

عند تطوير تطبيق ما يواجه المبرمج مشاكل وأخطاء تظهر أثناء التنفيذ أو الترجمة وتنقسم هذه الأخطاء إلى عدة أنواع: خطأ بالصيغة وهذا يسهل اكتشافه حيث لن يقوم الـ Compiler بترجمة المشروع وتنفيذه إن وجد خطأ من هذا النوع وقد تعترض عليه بيئة التطوير أثناء كتابتك لشفرة البرنامج - خط أحمر تحت العبارة - وأخطاء وقت التنفيذ وهذه أخطاء طارئة تحدث أثناء تنفيذ البرنامج ويجب مراقبتها في الشيفرة وهنا نستخدم عبارة Try ... Catch لحصر تلك الأخطاء وتجاوزها مثل عندما يحاول البرنامج فتح ملف قد يكون غير متوفر للفتح لأسباب متعددة مرتبطة ببيئة التشغيل ونوع آخر وهي أخطاء منطقية في الكود حيث تلاحظ أن صيغة الأوامر صحيحة ولكن البرنامج لا يقوم بالعمل كما يجب ففي هذه الحالة طرق تجاوز النوعين السابقين من المشاكل لن تفيدك وستضطر لاستخدام أدوت التنقيح Debugging tools لحصر وتصحيح تلك المشاكل وفيما يلي بعض النقاط التي تساعدك على استخدام هذه الأدوات لتجاوز المشاكل من النوع الأخير

يمكنك وضع نقاط التوقف Break Points لإيقاف تنفيذ البرنامج عند سطر معين ويمكن بعد التوقف متابعة تنفيذ البرنامج باستخدام F11 للمتابعة سطر سطر أو F5 لمتابعة تنفيذ البرنامج حيث يمكن وضع نقاط التوقف أو إزالتها باختيار البند Toggle Breakpoints من قائمة Debug أو ضغط المفتاح F9 أو النقر على الهامش الرمادي بجانب السطر المراد التوقف عنده وتظهر دائرة حمراء بجانب السطر دلالة على وضع نقطة التوقف عنده

لتشغيل البرنامج مع التنقيح اختر Start Debugging من قائمة Debug أو اضغط F5 ولتشغيله بدون تنقيح اختر Start Without Debugging أو اضغط Ctrl+F5

يمكنك ضغط المفتاح F11 لبدء البرنامج مع التنقيح سطر سطر

اضغط F11 سترى أنك قد انتقلت لأول سطر كود سيتم تنفيذه ولمتابعة تنفيذ البرنامج سطر سطر تابع ضغط F11 سترى في كل مرة أنه قد نفذ سطرا آخر من البرنامج حيث يمكنك استخدام هذه الطريقة للفهم الدقيق لكيفية تنفيذ البرنامج كما أن F10 تقوم بنفس عمل F11 تقريبا إلا أنها إذا واجهت إجراء ضمن الكود الذي يتم تنقيحه فإنها تمرر التنفيذ للسطر التالي فورا دون المرور بتنفيذ ذلك الإجراء سطر سطر كما تفعل F11 التي تنتقل لذلك الإجراء وتنفذه سطر سطر قبل العودة لتنفيذ باقي الكود المستدعي للإجراء

يمكنك إيقاف تنفيذ البرنامج وذلك إما بالضغط على زر التوقف من شريط الأدوات أو Shift-F5

اضغط F5 لتشغيل البرنامج وبهذا يبدأ تشغيل المنقح ويستمر تنفيذ الكود حتى يمر على نقطة توقف Break Point وعندها يتوقف عند السطر المحدد بنقطة التوقف المحددة سابقا وبينما أنت في وضع التوقف يمكنك متابعة بيانات الفئات في البرنامج عبر نافذتي Autos و Locals

نافذة locals تريك جميع المتغيرات المعرفة ضمن مجال التنفيذ الحالي حيث يمكنك استخدامها لرؤية جميع خصائص تلك المتغيرات وقيمها ونافذة Autos تعمل بطريقة مشابهة ولكنها ترينا متغيرات قد لا تكون معرفة ضمن مجال التنفيذ الحالي

إذا أوقفت مؤشر الفأرة فوق متغير أو خاصية ما وأنت في وضع التوقف ستلاحظ ظهور نافذة صغيرة تظهر لك تلك الخاصية وقيمتها ويمكنك عند الحاجة تغيير تلك الخاصية بالنقر المزدوج عليها وكتابة قيمة جديدة أو الضغط بزر الفأرة اليميني عليها ثم اختيار Edit Value من القائمة وتغيير تلك القيمة حيث يمكنك بعدها متابعة التنقيح باستخدام F11

لتغيير السطر التالي الذي سيتم تنفيذ الكود عنده فقط انقر بزر الفأرة اليميني على الخاصية واختر من القائمة Set Next Statement ستلاحظ تغير مكان السهم الأصفر الذي يدل على السطر التالي الذي سيتم تنفيذه

عندما توقف مؤشر الفأرة في وضع التوقف فوق نوع بيانات مركب مثل Me التي تشير للفئة الحالية مثلا أو متغير يشير إلى فئة أو تركيب ما أو قد يشير إلى Dataset مثلا يمكنك بالضغط على إشارات + لتنقل ورؤية جميع خصائص تلك الفئة أو نوع البيانات المركب أو تغييرها وذلك بنفس الطريقة التي تستخدمها للتنقل بين عناصر TreeView

إذا أردت تنفيذ البرنامج حتى يصل لسطر معين يمكنك فعل ذلك مباشرة بدون الضغط على F11 للتنفيذ وذلك بالضغط بزر الفأرة اليميني على ذلك السطر واختيار Run to Cursor حيث سيتم تنفيذ البرنامج حتى ذلك السطر

لمراقبة قيمة متغير بشكل مستمر نستخدم Watch window حيث يمكنك النقر بزر الفأرة اليميني على ذلك المتغير واختيار Add Watch حيث يمكنك رؤية ذلك المتغير ورؤية قيمته أو تغييرها مباشرة من تلك النافذة و بنفس الطريقة يمكنك أيضا إضافة Watch لأحد العناصر المركبة ورؤية أو تغيير قيمة إحدى خصائصه

لإزالة متغير من نافذة Watch فقط انقر بزر الفأرة اليميني عليه في تلك النافذة واختر Delete Watch

كما يمكنك كتابة اسم المتغير مباشرة في نافذة watch لمراقبته

في حالة وجود كمية بيانات كبيرة أو بنية بيانات معقدة داخل المتغير كبيانات XML مثلا يمكنك ملاحظة أيقونة مكبرة بجانب تلك القيمة حيث يمكنك إما الضغط على المكبرة مباشرة لعرض البيانات أو النقر على السهم الصغير بجانبها لاختيار طريقة عرض تلك البيانات من القائمة حيث يمكنك اختيار Xml Visualizer مثلا في حالة بيانات من نوع XML

يمكنك استخدام نقاط التعقب Trace Points ليقوم المنقح بتنفيذ عمل معين عند وصوله لهذه النقطة دون إيقاف تنفيذ البرنامج أو مع إيقاف التنفيذ

لوضع نقطة تعقب Tracepoint انقر بزر الفارة اليميني على سطر الكود ثم من القائمة الفرعية Breakpoint اختر Insert Tracepoint وهذا يؤدي إلى ظهور مربع حوار When Breakpoint Is Hit الذي يمكنك من تحديد ماذا تريد أن يفعل عندما يصل التنفيذ لذلك السطر حيث يوفر لك إمكانية طباعة رسالة أو تنفيذ ماكرو بالإضافة إلى خيار لاستمرار التنفيذ أو إيقافه عند ذلك السطر كما يمكنك استخدام تعابير معينة لإظهار قيم خاصة في سطر الرسالة مثل $TICK لإظهار استخدام المعالج أو $TNAME لإظهار اسم مسار التنفيذ الحالي Current Thread Name وعند ضبطها ستلاحظ ظهور معين أحمر بجانب السطر دلالة على Trace Point عوضا عن الدائرة الحمراء التي تشير لـ Break Point وستظهر الرسائل المتعلقة بـ Trace Point في نافذة Output

الاستثناءات Exceptions اصطياد الأخطاء ومعالجتها

التقاط استثناء معين

لالتقاط استثناء نستعمل بلوك Catch ... Try بشكل عام عندما ينفذ البرنامج عملية معينة قد تولد استثناء فلعمل ذلك نقوم بوضع تلك الشيفرة البرمجية بين عبارتي Try و Catch و بعد العبارة Catch نستكشف الاستثناءات الحاصلة

```
Try
      C = A + B
Catch Ex as OverflowException
```

ويتيح الجزء Catch للبرنامج اكتشاف استثناء معين والرد عليه فمثلا يمكننا التقاط استثناء القسمة على صفرDivided By Zero

```
Try
      C = A Mod B
      TextBox3.Text = C.ToString()
Catch Ex as DividedByZeroException
      MsgBox("Devided By Zero.")
      TextBox3.Text = "Infinity"
End Try
```

وبنفس الطريقة يمكننا استكشاف استثناء فيضان Overflow Exception

```
Dim A, B, C As Integer

Try
      A= TextBox1.text
      B = TextBox2.Text

      C = A + B
Catch Ex as OverflowException
      MsgBox("Overflow.")
      TextBox3.Text = "Infinity"
End Try
```

وحتى أيضا يمكننا استخدامه للكشف عن اسم ملف غير صالح

```
Dim Fname As New String = "D:\Some Folder\FileName.ext"
Dim Sfl as new StreamReader
Try
      Sfl = New StreamReader(Fname)
      TextBox1.Text = Sfl.ReadToEnd()
      Sfl.Close
Catch Ex As FileNotFoundException
      MsgBox("File Not Found")
End Try
```

فحص عدة استثناءات

عندما يمكن أن تؤدي العملية التي تقوم بتنفيذها إلى عدة استثناءات مختلفة يمكنك تحديد سلسلة من الجمل Catch لمعالجة تلك الاستثناءات

```
Dim FileDB As New OpenFileDialog()
```

```
FileDB.Filter = "All files | *.* | Text files | *.txt"

FileDB.FilterIndex = 2
FileDB.InitialDirectory = "C:\Temp"
FileDB.AddExtension = True
FileDB.DefaultExt = "txt"

' Prevent dialog box from validating file
FileDB.CheckFileExists = False
FileDB.CheckPathExists = False

If (FileDB.ShowDialog() = DialogResult.OK) Then
      Dim SourceFile As StreamReader

      Try
            SourceFile = New StreamReader(FileDB.FileName)

            TextBox1.Text = SourceFile.ReadToEnd()
                  SourceFile.Close()
      Catch Except As DirectoryNotFoundException
            MsgBox("Error: " & Except.Message)
      Catch Except As FileNotFoundException
            MsgBox("Error: " & Except.Message)
      Catch Except As Exception
            MsgBox("Error: " & Except.Message)
      End Try
Else
      MsgBox("User selected Cancel")
End If
```

معالجة الاستثناءات باستعمال بلوك Catch عام

عندما ينفذ كائن عملية نيابة عن البرنامج فقد يولد نطاقا واسعا من الاستثناءات بناء على سير تنفيذ البرنامج وقد لا تهمك ما هي هذه الاستثناءات بقدر ما يهمك أنه قد حصل هناك استثناء ما ولمعالجة الاستثناءات بغض النظر عن نوعها فإننا لا نحدد استثناء معينا بل نستخدم

```
Catch Ex as Exception
```

مثال

```
Try

      ..... Some Code Here

Catch Ex As Exception
      MsgBox("Error: " & Ex.Message)
End Try
```

إجراء التنظيف بعد حدوث استثناء

عند استعمالك لبلوك Try ... Catch للرد على سلسلة من الاستثناءات ستنفذ عادة عمليات تخص كل استثناء ضمن بلوك Catch المناسب وبناء على الأمور التي يقوم بها برنامجك عليك القيام بعمليات معينة بعد حدوث استثناء وذلك بغض النظر عن نوع الاستثناء ولهذا الغرض نستخدم عبارة Finally في نهاية بلوك Try ... Catch تحدد الجمل التي نريد تنفيذها بغض النظر عن نوع الاستثناء مع ملاحظة أن العبارات الموجودة ضمن بلوك Finally سيتم تنفيذها دوما بغض النظر عن حدوث استثناء أو لا

```
Dim FileDB As New OpenFileDialog()
```

```
FileDB.Filter = "All files | *.* | Text files | *.txt"

FileDB.FilterIndex = 2
FileDB.InitialDirectory = "C:\Temp"
FileDB.AddExtension = True
FileDB.DefaultExt = "txt"

' Prevent dialog box from validating file
FileDB.CheckFileExists = False
FileDB.CheckPathExists = False

If (FileDB.ShowDialog() = DialogResult.OK) Then
      Dim SourceFile As StreamReader

      Try
            SourceFile = New StreamReader(FileDB.FileName)
      Catch Except As Exception
            MsgBox("Error: " & Except.Message)
      End Try

      If (Not SourceFile Is Nothing) Then
            Try
                  TextBox1.Text = SourceFile.ReadToEnd()
            Catch Except As Exception
                  MsgBox("Error: " & Except.Message)
            Finally
                  MsgBox("In finally statements")
                  SourceFile.Close()
            End Try
      End If
Else
      MsgBox("User selected Cancel")
End If
```

وفي بعض الحالات قد تكون هناك أوقات لا تريد استكمال تنفيذ البلوك Catch ... Try عندها تستخدم العبارة Exit Try للخروج من البلوك حيث سينفذ بعدها أول سطر كود يلي End Try

```
Try

      .... Some Code

      If SomeCondition Then Exit Try

Catch Ex as Exception

  Exit Try

  .... Rest of Try Block
```

إطلاق استثناءاتك الخاصة

هناك أوقات تحتاج فيها لتكوين استثناء خاص بك عندها ستحتاج بكل بساطة لإنشاء فئة Class ترث الفئة - Exception فعلى سبيل المثال يمكننا توليد استثناء باسم InvalidEMailException كما يلي

```
Public Class InvalidEMailException
```

```
        Inherits System.Exception

        Sub New(ByVal Message As String)
              MyBase.New(Message)
        End Sub
End Class
```

و بالطبع يمكنك إنشاء طرق وخصائص في هذه الفئة حسب احتياجاتك كأي فئة أخرى وفي مثالنا المبسط هنا أنشأنا مشيد الفئة فقط وبعد إنشاء فئة الاستثناء الخاصة بنا يمكننا توليد الاستثناء باستخدام العبارة Throw

```
Throw New InvalidEMailException("Envalid Email Please Correct")
```

وفيما يلي مثال آخر

```
Public Class MyException
        Inherits System.Exception

        Sub New(ByVal Message As String)
              MyBase.New(Message)
        End Sub
End Class

Public Class Form1
        Inherits System.Windows.Forms.Form

        .....

        Private Sub Button1_Click(ByVal sender As Object, _
                    ByVal e As System.EventArgs) Handles Button1.Click

              Try
                    MsgBox("About to generate custom exception")
                    Throw (New MyException("** Custom Message **"))
              Catch Ex As MyException
                    MsgBox("Custom Exception thrown " & Ex.Message)
              End Try

        End Sub

        ......

End Class
```

ترشيح الأخطاء في قسم CATCH في بلوك TRY عند اصطياد الأخطاء

يوفر لنا قسم Catch في بلوك Try أكثر من خيار لترشيح الأخطاء وإحدى هذه الطرق هي بتحديد نوع الخطأ المراد اصطياده وهنا يجب عليك البدء بالنوع الأكثر تحديدا منتهيا بالنوع الأكثر عمومية بما أن قسم Catch يتم تنفيذه بترتيب كتابته كما يمكن استخدام When في قسم Catch لتحديد أكثر دقة مثل تحديد رقم خطأ معين حيث يمكنك دمج هذه الأساليب للحصول على الطريقة المناسبة لبرنامجك
مثال:

```
Try
     ' خطأ يسبب ان المحتمل كودك
Catch ex As System.IO.IOException
     ' كود للتفاعل مع الخطأ IOException
Catch ex As System.NullReferenceException
     ' كود للتفاعل مع الخطأ NullReferenceException
Catch ex As Exception When Err.Number = 5 Or Err.Number = 8
     ' كود للتفاعل عندما يكون رقم الخطأ 5 أو 8
Catch ex As Exception
     ' كود للتفاعل مع أي خطأ آخر
```

```
End Try
```

فيما يلي بعض فئات الأخطاء ووصف سريع لكل منها

الفئة	الوصف
AmbiguousMatchException	لم يستطع البرنامج تحديد أي نسخة من الدالة يستخدم
ApplicationException	هذه هي الفئة الأب لجميع فئات الأخطاء الغير قاتلة. فعندما تبني فئة استثناء خاصة بك يجب أن تكون موروثة من هذه الفئة
ArgumentException	القيمة الممررة غير صحيحة
ArgumentNullException	القيمة الممررة لا يمكن أن تكون لاشئ ومع ذلك قيمتها لاشئ
ArgumentOutOfRangeException	القيمة الممررة خارج المجال المقبول
ArithmeticException	خطأ إسناد أو تحويل في عملية حسابية
ArrayTypeMismatchException	البرنامج يحاول القيام بإدخال عنصر من نوع خاطئ في المصفوفة
ConfigurationException	قيمة الإعداد غير صحيحة
ConstraintException	عملية البيانات تسبب خطأ في القيود على قاعدة البيانات
DataException	الفئة الأب لجميع فئات الأخطاء المتعلقة بـ ADO .NET
DirectoryNotFoundException	المجلد المطلوب غير موجود
DivideByZeroException	خطأ القسمة على صفر
DuplicateNameException	عملية ADO .Net واجهت اسما مكررا. مثل أنك تحاول إنشاء جدول مع أنه يوجد جدول موجود سابقا ويملك نفس الاسم
EvaluateException	يحدث عندما لا يستطيع البرنامج تقييم قيمة التعبير الموجود في عمود قاعدة البيانات
FieldAccessException	البرنامج يحاول الوصول إلى خاصية للفئة بطريقة غير صحيحة
FormatException	تنسيق القيمة الممررة غير صحيح
IndexOutofRangeException	البرنامج يحاول الوصول إلى عنصر يقع خارج حدود المصفوفة أو أي عنصر احتواء آخر
InvalidCastException	البرنامج يحاول القيام بتحويل نوع غير صحيح
InvalidOperationException	العملية المطلوبة حاليا غير مسموح بها
IOException	الفئة الأب لجميع فئات أخطاء الدخل/الخرج. خطأ دخل/خرج عام
EndOfStreamException	وصل الـ Stream إلى نهايته
FileLoadException	خطأ أثناء تحميل الملف
FileNotFoundException	لا يمكن إيجاد الملف المطلوب
InternalBufferOverflowException	حدوث فيضان في الـ Buffer الداخلي
MemberAccessException	البرنامج يحاول الوصول إلى عنصر في فئة بطريقة غير صحيحة
MethodAccessException	البرنامج يحاول الوصول إلى الطريقة بطريقة غير صحيحة
MissingFieldException	البرنامج يحاول الوصول إلى خاصية غير موجودة في الفئة
MissingMemberException	البرنامج يحاول الوصول إلى عنصر غير موجود في الفئة
MissingMethodException	البرنامج يحاول الوصول إلى طريقة غير موجودة في الفئة
NotImplementedException	العملية المطلوبة غير معرفة
NotSupportedException	الخاصية المطلوبة غير مدعومة
NullReferenceException	البرنامج يحاول استخدام مرجع إلى غرض Object قيمته لاشئ
OutOfMemoryException	لاتوجد ذاكرة كافية فمثلا إن كان المستخدم يحاول توليد مجموعة بيانات ضخمة يمكنك التنبؤ بحجم الذاكرة التي سيحتاجها البرنامج واختبار إن كانت متوفرة فتقوم برمي هذا الاستثناء إن لم تكن كافية
OverflowException	خطأ فيضان في عملية حسابية
RankException	الإجراء يحاول استخدام مصفوفة تملك عددا خاطئا من الأبعاد
ReadOnlyException	البرنامج يحاول تعديل بيانات للقراءة فقط
ResourceException	المصدر المطلوب مفقود
SyntaxErrorException	خطأ في الصيغة عند إسناد قيمة لخاصية
UnauthorizedAccessException	النظام يمنع الوصول بسبب عدم كفاية الصلاحيات

التقاط الاستثناءات الغير معالجة في التطبيق

في Application Events يوجد الحدث UnhandledException الذي يستقبل جميع الاستثناءات الغير معالجة في التطبيق مع ملاحظة أنه عندما يصل الاستثناء لهذا الحدث فإن التطبيق سيتم إنهاؤه و لا يعود مسار التنفيذ لداخل التطبيق

القسم الثاني - البرمجة المتعلقة بـ VBA و VB6 و Microsoft Office

ويضم المواضيع التالية:

- الفروقات بين VB2008 و #C و VB6
- ملاحظات هامة عند ترقية مشاريع VB6 إلى VB 2008
- مكتبة التوافقية الخاصة بفيجول بايزيك 6
- هل تعاني من مشكلة في معالج تحديث الكود من VB6 إلى VB2005
- استخدام كود فيجول بيزيك دوت نيت في فيجول بيزيك 6
- كيف يمكننا استخدام فيجول بايزيك 2008 لإنشاء صفحات أشرطة إضافية لـ Excel 2007
- أتمتة وورد 2007 باستخدام فيجول بايزيك دوت نيت
- كتابة شيفرة لإنشاء Add-in يتم استدعاؤه من VBA
- كيف نستدعي صناديق الحوار الخاصة بمايكروسوفت وورد من برنامجنا

الفروقات بين VB2008 و C#3.0 و VB6

هذه الإصدارة من فيجول بايزيك قد تم تصميمها خصيصا لتستخدم نماذج البرمجة الجديدة التي تم تقديمها ضمن الفريموورك 3.5 وقد تم تصميم كلا من VB و C# للعمل مع مكتبات زمن التشغيل للفريموورك CLR حيث تم تصميم لغة C# لاستهداف جمهور المبرمجين الذين يعملون على C++ بينما تم تصميم الفيجول بايزيك 2008 لاستهداف الجمهور العريض لمبرمجي البايزيك السابقين ومن أجل أن تتمكن لغة البيزيك من استخدام الميزات المقدمة من قبل الفريموورك 3.5 كان لابد من أن تقدم مزيدا من الدعم الأقوى للبرمجة غرضية التوجه OOP مع احتفاظها بميزات أمان الأنواع

بين VB2008 و C#

تعتبر هاتان اللغتان متطابقتان فيما يتعلق بما يمكنك تحقيقه بهما حيث يمكن استخدام أيا منهما للوصول إلى جميع الفئات والوظائف المقدمة من قبل الفريموورك وبشكل أساسي يمكنك أن تطور أي شيئ باستخدام VB2008 تماما كما لو أنك طورته باستخدام C# مع أن إحدى هاتين اللغتين قد تقدم لك انسيابية أكثر اعتمادا على ما تقوم بعمله وإن أردنا المقارنة بين اللغتين سنجد أنه من الأسهل لنا أنت نتحدث عن الفروقات من أن نتحدث عن الأشياء المشتركة بينهما فكلتاهما تدعمان جميع الإضافات الجديدة للغة مثل LINQ و الأنواع المجهولة وتعابير لمدا وغيرها ويكمن الفرق الحقيقي في أن C#3.0 تقدم إمكانية لكتابة كود غير آمن بينما تقدم VB إمكانية الربط المتأخر حين يكون الكود الغير آمن في C# عبر استخدام المؤشرات لإدارة الذاكرة مباشرة فربما تحتاج الكود الغير آمن لأسباب تتعلق بالأداء أو لبعض استدعاءات Windows API ذات المستوى المنخفض حيث يكون الكود الغير آمن مفيدا في بعض الحالات مع عدم النصح باستخدامه لأنه لا يمكنك التأكد من أمانه إضافة إلى أنك لن تستطيع التأكد من أن جامع النفايات سيتمكن من إزالة الأغراض الموجودة في الذاكرة وتحرير مصادر النظام المستخدمة من قبل تلك الأغراض بينما تم الإبقاء على الربط المتأخر في لغة الفيجول بايزيك من أجل التوافقية مع النسخ السابقة حيث أن الربط المتأخر يسمح لك بإنشاء متغيرات من النوع OBJECT ثم تعيينها بواسطة متغير من نوع ما أو باستخدام الدالة CreateObject ويفضل الربط المبكر عن الربط المتأخر لأنه يرفع مستوى الأداء إضافة إلى أن استخدام الربط المتأخر لا يمكن المترجم من اكتشاف بعض الأخطاء مما يعني إمكانية كبيرة لحدوث أخطاء في زمن التشغيل

بين VB2008 و VB6

تم إعادة صياغة لغة الفيجول بايزيك بشكل كلي ابتداء من VB2002 وذلك من أجل دعم CLR فهي تحمل شبها اسميا فقط مع VB6 مما عنى انعطافا كبيرا في التعلم بالنسبة للعديد من المبرمجين ولكن بالمقابل عندما تطبق الطريقة البرمجية الجديدة بشكل صحيح باستخدام بيئة التطوير المحسنة ينتج لديك برمجيات أفضل ويقدم لنا CLR إضافة لذلك إدارة محسنة للذاكرة وأمان الأنواع والبرمجة غرضية التوجه OOP وأحد التغييرات الكبرى التي طرأت على لغة البيزيك هي أنها أصبحت لغة حقيقية للبرمجة غرضية التوجه OOP مما يعني أن جميع الأغراض هي موروثة من System.Object وبدلا من VB Runtime و Win32 API أصبح لدينا مكتبة فئات أساسية كاملة BCL لتعمل معها ويكمن التحدي للمبرمجين المنتقلين حديثا للإصدارة الجديدة في الإبحار عبر الفريموورك ومعرفة أين يجدون الفئات التي يحتاجونها وكلما أصبحت متآلفا أكثر مع الفريموورك أصبح بإمكانك كتابة برامج أفضل وأسرع حيث يعتبر BCL جزءا من الفريموورك يوفر أنواعا تستخدم في أي كود سيتم كتابته للعمل ضمن CLR وهذا يتضمن العديد من مجالات الأسماء مثل System.Collections و System.IO و System.Registry و System.Text و System.Drawing وغيرها

ملاحظات هامة عند ترقية مشاريع VB6 إلى VB .net 2008

على الرغم من أن عملية ترقية مشاريع VB6 إلى إصدار 2008 تتم بمعظمها بصورة آلية إلا أنه هناك بعض النقاط الواجب أخذها بعين الاعتبار للتحضير لعملية الترقية. وبمراعاة الملاحظات التي سترد هنا يمكنك تقليل أو إلغاء العديد من التعديلات التي ستضطر لعملها بعد انتهاء معالج الترقية وفي معظم الحالات تكون هذه التوصيات عبارة عن ممارسات برمجة جيدة وسيفيدك معرفة الطرائق والأغراض التي لا يوجد لها مكافئات في عملية الترقية. وبشكل عام فإن لم تتم عملية ترجمة وتشغيل المشروع بصورة جيدة ضمن بيئة VB6 فلا تتوقع أن تتم عملية الترقية بنجاح لهذا يقترح تنصيب VB6 على الجهاز الذي ستتم عليه عملية الترقية واختباره هناك أولا. إضافة إلى أن معالج الترقية إلى 2008 يقوم بالترقية من الإصدار 6 فقط فإن كانت لديك مشاريع على الإصدارات من 1 إلى 5 فيجب فتحها ضمن بيئة تطوير VB6 وترقيتها إلى الإصدار السادس قبل البدء بعملية الترقية للإصدار 2008 ولا تنس استخدام الخيار upgrade Microsoft ActiveX controls عند ترقية مشاريع الإصدارات القديمة للإصدار السادس.
يمتلك الإصدار 2008 رزمة من النوافذ وتحكماتها متوافقة بشكل كبير مع تلك الموجودة في الإصدار السادس ومع ذلك توجد بعض الاختلافات التي سنوردها:

- الإصدار 2008 لا يدعم التحكم Ole Container لذا يجب عليك تجنب استخدامه في المشاريع التي تنوي ترقيتها
- لا يوجد Shape Control في الإصدار 2008 حيث سيتم ترقية المربعات والمستطيلات إلى تحكمات Label بينما الدوائر والأشكال الاهليلجية لا يمكن ترقيتها لذا يجب عليك تجنب استخدامها
- لا يوجد Line Control في الإصدار 2008 حيث سيتم ترقية الخطوط الأفقية والشاقولية إلى تحكمات Label بينما الخطوط المائلة لن يتم ترقيتها لذا يجب عليك عدم استخدامها
- يمتلك الإصدار 2008 أوامر رسومية جديدة تستبدل الطرائق التالية لـ Form وهي Circle و Cls و PSet و Line و Point وبسبب أن الـ Object Module الجديدة مختلفة عن القديمة لذا لا يمكن ترقية هذه الطرائق
- من أجل التحكم Timer عند ضبط الخاصية Interval إلى الصفر فإن التحكم لا يتوقف عمله بل سيتم إعادة ضبط قيمة الخاصية إلى 1 واحد ولإيقاف عمل التحكم Timer يجب عليك ضبط الخاصية Enabled إلى False عوضا عن ضبط قيمة Interval إلى الصفر
- يمتلك الإصدار 2008 تحكمان خاصان بالقوائم هما MenuStrip و ContextMenuStrip بينما في الإصدار السادس هناك تحكم قوائم واحد يمكن استخدامه كـ Menu أو ContextMenu لذا سيتم ترقية جميع تحكمات القوائم إلى تحكم MenuStrip يمتلك عدة MenuItems من أجل كل تحكم قائمة وعند ترقية ContextMenu يجب عليك إعادة إنشائها وحذف التحكمات MenuStrip الزائدة
- لا يمتلك الإصدار 2008 دعما لـ Dynamic Data Exchange DDE
- على الرغم من أن الإصدار 2008 يمتلك دعما لوظيفة Drag And Drop إلا أنها تختلف بشكل كبير عن تلك الموجودة في الإصدار السادس لذا فإن وظائف السحب والإفلات لا يمكن ترقيتها
- يمتلك الإصدار 2008 دعما محسنا للغرض Clipboard من خلال My.Computer.Clipboard حيث يقدم دعما لوظائف وصيغ أكثر من تلك الموجودة في الإصدار السادس وبسبب الاختلافات الكبيرة فلا يمكن القيام بعملية ترقية الكود المستخدم لـ Clipboard بصورة آلية
- لا يدعم الإصدار 2008 الخاصية Name للنموذج والتحكمات في زمن التشغيل لذا يجب عليك الدوران من خلال Controls collection عند بحثك عن تحكم يمتلك اسما معينا. وهذه الوظيفة متوفرة في .net Framework من خلال الفئات System.Reflection
- وفيما يتعلق بقواعد البيانات فإن الإصدار 2008 يمتلك نسخة محسنة من ADO هي ADO .net محسنة من أجل البيانات في التطبيقات الموزعة. وعلى الرغم من أنه يمكنك استخدام RDO و ADO في الإصدار 2008 مع بعض التعديلات البسيطة إلا أنه لا يدعم أدوات الربط مع قواعد البيانات الخاصة بـ DAO و RDO وأيضا Data Controls و لا حتى RDO User Connection. لذا ينصح أنه إذا كان برنامجك يستخدم تحكمات DAO و RDO بإما تركهم في الإصدار السادس أو ترقيتهم إلى تحكمات ADO قبل القيام بعملية الترقية للإصدار 2008 بما أن Windows Forms يدعم تحكمات ADO

سؤال

في حالة الترقية من Vb.net 2002 أو 2003 إلى Vb.net 2008 هناك عملية ترقية تلقائية هل توجد شروط أو حالات معينة يجب التنبه إليها في حالة الترقية

الجواب

بالنسبة لمشاريع 2002 و 2003 سيتولى معالج الترقية معظم العملية ويتبقى عليك مراجعة الكود لمتابعة الملاحظات التي قد يضعها معالج الترقية وإجراء بعض التصحيحات البسيطة ضمن الكود وبشكل عام ستتم عملية الترقية بسلاسة وبالنسبة لمشاريع 2005 ستتم عملية الترقية بسلاسة ودون أي مشاكل تذكر

هناك ملاحظة تتعلق بموضوع استقرار الكود نتيجة التجربة تستحق الذكر

عندما نستخدم إجراء من مكتبة خارجية user32 وليكن GetWindowLong في VB6 كنا نعرفه بالشكل

```
Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
          (ByVal hwnd As Long, ByVal nIndex As Long) As Long
```

وفي نسخة الـ 2002 و الـ 2003 كنا نستطيع استخدام نفس التعريف بدون اعتراض بيئة التطوير رغم أن مجال المتغيرات قد اختلف بين VB6و الـ net. فالنوع Long في vb6 يكافئ Integer في net. وبناء عليه فالتعريف السابق كان يجب تصحيحه إلى

```
Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" _
       (ByVal hwnd As Integer, ByVal nIndex As Integer) As Integer
```

وهنا ابتداء من الإصدارة 2005 أصبحت بيئة التطوير تولد خطأ في زمن التنفيذ عند التعامل مع تعريفات لإجراءات من مكتبات خارجية لم يتم تعريف أنواع المتغيرات فيها بدقة لذا يجب تصحيح هكذا تعريفات حتى يتم تنفيذ مشروعك بصورة صحيحة بعد عملية الترقية

كيف تتم ترقية المشاريع من VB6

- افتح البرنامج على VB6 واختبره جيدا وصحح النقاط الواجب تصحيحها وفق الملاحظات الواردة سابقا
- احفظ نسخة احتياطية من المشروع دوما قبل القيام بعملية الترقية
- افتح المشروع بواسطة بيئة تطوير VB2008 واتبع خطوات المعالج البسيطة

سؤال

- لدي قطعة كود على VB6 أو VBA وأريد ترقيتها ولكنها ليست مشروع كامل

الجواب

- افتح مشروعك الذي تنوي استخدام قطعة الكود فيه ثم انتقل في محرر الكود إلى النقطة التي ترغب في إدراج الكود بعد الترقية فيه
- من القائمة Tools اختر Upgrade Visual Basic 6 Code ألصق الكود المراد ترقيته في صفحة Code ثم أضف أية مراجع ربما يحتاجها الكود من الصفحة References ثم اضغط زر Upgrade وانتظر قليلا فتتم عملية ترقية الكود وإدراجه في محرر الكود لديك

مكتبة التوافقية الخاصة بفيجول بايزيك 6.0

يعتبر فيجول بايزيك 2008 تطورا كبيرا عن فيجول بايزيك 6 حيث يمكننا اعتباره لغة جديدة في عائلة لغات البايزيك كما يمكننا إيجاد العديد من الوظائف والتعدادات والأنواع المخصصة والأغراض التي كانت موجودة سابقا في فيجول بايزيك 6 في مكتبات فئات الدوت نيت. وبالنظر إلى هذه الحقيقة فإن إي مشروع فيجول بايزيك 2008 جديد يتضمن مرجعا تلقائيا إلى أحد مجمعات الدوت نيت المسمى Microsoft.VisualBasic.dll الذي يحدد أنواع تزودنا بوظائف موروثة من فيجول بايزيك 6 وكأي مجمع آخر في الدوت نيت فإن Microsoft.VisualBasic.dll مؤلف من العديد من مجالات الأسماء المجمعة مع بعضها.
وتكون هذه المجمعات متوفرة تلقائيا لكل ملف vb في مشروعك وهذا يعني أنك لا تحتاج إلى تصريح الاستيراد Import للوصول إلى تلك الأنواع. وبهذا مازال يمكنك الاستفادة من العديد من عناصر فيجول بايزيك 6 مثل الوظيفة MsgBox التي تستدعى لإظهار صندوق رسائل بسيط كما في المثال

```
' The Microsoft.VisualBasic namespaces
' are automatically referenced by a
' Visual Studio 2008 VB project
Module Module1
        Sub Main()
                MsgBox("Hello, old friend …")
        End Sub
End Module
```

والطريقة MsgBox هي عنصر في Module في فيجول بايزيك 2008 تسمى Interaction وهي معرفة ضمن مجال الأسماء Microsoft.VisualBasic. وستلاحظ أن الـ Module في فيجول بايزيك 2008 تماثل ملف bas في فيجول بايزيك 6 في أن العناصر المحتواة ضمنها يمكن استدعاؤهم بدون استخدام اسم الـ Module كبادئة ومع ذلك إن أردت استخدام اسم الـ Module كبادئة عند استخدام الطريقة MsgBox يمكن أن يصبح كودنا السابق كما يلي

```
Module Module1
        Sub Main()
                Interaction.MsgBox("Hello, old friend …")
        End Sub
End Module
```

ومع أننا نشعر بالاطمئنان لمعرفة أن وظائف فيجول بايزيك 6 مازال يمكن تمثيلها ضمن مشاريع فيجول بايزيك 2008 فينصح هنا بتجنب استخدام هذه الأنواع قدر الإمكان نظرا لأن مايكروسوفت تخطط الإزالة دعم فيجول بايزيك 6 مع الزمن وأنه لا يمكنك ضمان أن مايكروسوفت ستوفر هذا المجمع مستقبلا. وتوفر مكتبات الفئات الأساسية العديد من الأنواع المدارة التي تقدم وظائف أكثر من تلك الموجودة في المكتبة الموروثة من لغة فيجول بايزيك 6. ويجب أن تتعلم كيف تقوم بالعمل بدون استخدام موجودات مكتبة التوافقية مع فيجول بايزيك 6 والقيام بالأمور باستخدام فئات الدوت نيت.

هل تعاني من مشكلة في معالج تحديث الكود من VB6 إلى VB2005

ظاهرة المشكلة

المعالج لا يتم تنفيذه ويظهر لك خطأ رقم 0x800706BE أو 0x800706BA أو كليهما

أقدم إليك مجموعة من الحلول التي ممكن أن تفيد في تجاوز هذه المشكلة

الحل الأول:

إذا كنت قد وضعت إعداد اللغة الافتراضية إلى أحد اللغات الموجودة في الجدول التالي:

10241	0x2801	Arabic (Syria)	ar-SY	1256	ARS
10249	0x2809	English (Belize)	en-BZ	1252	ENL
10250	0x280a	Spanish (Peru)	es-PE	1252	ESR
11265	0x2c01	Arabic (Jordan)	ar-JO	1256	ARJ
11273	0x2c09	English (Trinidad)	en-TT	1252	ENT
11274	0x2c0a	Spanish (Argentina)	es-AR	1252	ESS
12289	0x3001	Arabic (Lebanon)	ar-LB	1256	ARB
12297	0x3009	**Windows 98/Me, Windows 2000 and later:** English (Zimbabwe)	en-ZW	1252	ENW
12298	0x300a	Spanish (Ecuador)	es-EC	1252	ESF
13313	0x3401	Arabic (Kuwait)	ar-KW	1256	ARK
13321	0x3409	**Windows 98/Me, Windows 2000 and later:** English (Philippines)	en-PH	1252	ENP
13322	0x340a	Spanish (Chile)	es-CL	1252	ESL
14337	0x3801	Arabic (U.A.E.)	ar-AE	1256	ARU
14346	0x380a	Spanish (Uruguay)	es-UY	1252	ESY
15361	0x3c01	Arabic (Bahrain)	ar-BH	1256	ARH
15370	0x3c0a	Spanish (Paraguay)	es-PY	1252	ESZ
16385	0x4001	Arabic (Qatar)	ar-QA	1256	ARQ
16394	0x400a	Spanish (Bolivia)	es-BO	1252	ESB
17418	0x440a	Spanish (El Salvador)	es-SV	1252	ESE
18442	0x480a	Spanish (Honduras)	es-HN	1252	ESH
19466	0x4c0a	Spanish (Nicaragua)	es-NI	1252	ESI
20490	0x500a	Spanish (Puerto Rico)	es-PR	1252	ESU
31748	0x7c04	Chinese (Traditional)	zh-Hant		

فيجب عليك تعديل اللغة الافتراضية من لوحة التحكم إلى أي لغة أخرى وبالنسبة لمستخدمي اللغة العربية يمكنهم وضع اللغة على Arabic (Saudi Arabia)

الحل الثاني

ربما يكون الملف VBUpgrade.Engine.DLL غير مسجل أعد تسجيله يدويا باستخدام الأمر

```
regasm  C:\Program Files\Microsoft Visual Studio 8\VB\VBUpgrade\VBUpgrade.Engine.DLL
```

الحل الثالث

ربما يكون الملف Microsoft.VisualBasic.UpgradeExtensions.DLL غير مسجل أعد تسجيله يدويا مع العلم أنه موجود في الإصدارة 2005 في المسار

```
C:\Program Files\Microsoft Visual Studio 8\VB\VBUpgrade\
```

وذلك باستخدام الأمر

```
regasm  Microsoft.VisualBasic.UpgradeExtensions.DLL
```

الحل الرابع

تأكد من الريجستري إذا كان موجودا فيه المفتاح التالي قم بحذفه ثم أعد تسجيل جميع المكتبات الموجودة في المجلد C:\Program Files\Microsoft Visual Studio 8\VB\VBUpgrade\ بواسطة الأداة Regasm

```
[HKEY_CLASSES_ROOT\CLSID\{A8220117-B52C-4012-8CB7-2E0202B3A624}\InprocServer32\8.0.1200.0]

"Class"="Microsoft.VisualBasic.UpgradeExtensions.ResUtil"

"Assembly"="Microsoft.VisualBasic.UpgradeExtensions,Version=8.0.1200.0,Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"

"RuntimeVersion"="v2.0.40607"
```

استخدام كود فيجول بيزيك دوت نيت في فيجول بيزيك 6

يمكن للبرامج المبنية على COM مثل VB6 التعامل مع الكود الخاص بك والمكتوب ضمن الفريموورك مثل Class - Interface - struct - enum إذا تم إتباع القواعد التالية بشكل عام:

- يجب على الفئات تعريف واجهة
- التعريف باستخدام Public
- الأنواع Types لا يمكن أن تكون مجردة

كما يمكنك تحديد فيما إذا كان يمكن أو لا يمكن رؤية الإجراء أو الفئة أو ... الخ من عملية الـ com باستخدام الواصفة ComVisibleAttribute كالمثال

```
Imports System.Runtime.InteropServices

<ComVisible(False)> _
Class SampleClass

   Public Sub New()
       'Insert code here.
   End Sub

   <ComVisible(False)> _
   Public Function MyMethod(param As String) As Integer
       Return 0
   End Function

   Public Function MyOtherMethod() As Boolean
       Return True
   End Function

   <ComVisible(False)> _
   Public ReadOnly Property MyProperty() As Integer
       Get
           Return MyProperty
       End Get
   End Property

End Class
```

ويجب عليك تسجيل مجمع . net لـعملاء الـ Com حتى يمكن استخدامها وذلك باستخدام Tlbexp.exe و RegAsm.exe للقيام بعملية التسجيل ويستخدم من سطر الأوامر بالشكل التالي

```
TlbExp AssemblyName /out:FileName
```

و عملاء الـ com يصلون للمجمع الخاص بك عن طريق ملف .tlb المنشأ بواسطة هذه الأداة كما يجب عليك تسجيل المجمع الخاص بك عن طريق Regasm.exe والذي يمكنه توليد ملف tlb أيضا عن طريق الخيار /tlb: والذي يمكن استخدامه بالشكل

```
RegAsm AssemblyName /tlb: FileName.tlb /codebase
```

دعنا نقوم بعمل مثل عملي على الموضوع

افتح بيئة التطوير وابدأ مشروعا جديدا من نوع Class Library وسمه TestProj فيتم إنشاء Class1 افتراضيا ثم ادخل الكود التالي في Class1

```
Option Strict On

Public Class Class1
  Public Function myFunction() As Integer
     Return 100
  End Function
End Class
```

- افتح خصائص المشروع واضغط الزر Assembly Information من الصفحة Application وتأكد من وضع إشارة بجانب Make Assembly COM-Visible ثم اضغط OK
- من صفحة Compile تأكد من وضع إشارة بجانب Register for COM Interop
- قم بالحفظ ثم قم بعمل Build للمشروع
- افتح VB6 ثم قم بإنشاء مشروع افتراضي جديد
- من قائمة Project اختر References ومن Available References اختر اسم مشروعنا TestProj ثم اضغط Ok
- ضع زر أوامر على النموذج الخاص بمشروعك ثم انقر عليه نقرا مزدوجا لينقلك إلى محرر الكود و في الحدث Button1_Click أدخل الكود التالي

```
Dim myObject As TestProj.Class1
Set myObject = New TestProj.Class1
MsgBox myObject.myFunction
```

شغل مشروعك واختبره

كيف يمكننا استخدام فيجول بايزيك 2008 لإنشاء صفحات أشرطة إضافية لـ Excel 2007

- قم بإنشاء مشروع فيجول بايزيك جديد وذلك بتحديد Project Type إلى Office ثم 2007 ثم اختيار الـ Template هي Excel 2007 WorkBook
- من قائمة Project اختر Add New Item ثم اختر التحكم Actions Pane Control ثم اضغط Add
- من الـ toolbox أضف التحكمات التي تريدها لـ Actions Pane Controls مثلا Label وغير النص بداخله إلى Actions Pane 1
- كرر العملية وأضف تحكما آخر من النوع Actions Pane Control وضع عليه تحكم Label وغير النص بداخله إلى Actions Pane 2
- اختر Add New Item من قائمة Project و أضف تحكم من النوع Ribbon (Visual Designer) للمشروع بعد تغيير تسميته من Ribbon1 إلى MyRibbon ثم اختر Group1 من المحرر الذي يظهر لك وغير الخاصية Label إلى Actions Pane Manager
- من الـ ToolBox ومن قسم Office Ribbon controls أضف Button إلى داخل المجموعة Actions Pane Manager وغير الخاصية Label لتحمل القيمة Show Actions Pane 1 وأضف زرا آخر وغير خاصية Label له إلى Show Actions Pane 2 ثم أضف تحكما آخر لنفس المجموعة من نوع ToggleButton وغير خاصية Label إلى Hide Actions Pane
- يمكننا ملاحظة إمكانية إضافة صفحات إضافية لـ My Ribbon وذلك بالنقر بزر الماوس اليميني في الفراغ بجانب TabAddIns واختيار الأمر Add Ribbon Tab وإضافة الصفحات التي تريدها ثم وضع التحكمات عليها كما سنرى لاحقا جرب إضافة صفحة أو أكثر كما تشاء ثم عد إلى TabAddIns لمتابعة العمل معا
- من solution Explorer انقر بزر الماوس اليميني على MyRibbon ثم اختر الأمر View Code من قائمة السياق
- تحت تعريف الفئة مباشرة أدخل سطري الكود التاليين

```
    Dim ActionsPane1 As New ActionsPaneControl1
    Dim ActionsPane2 As New ActionsPaneControl2
```

وذلك لتعريف متغيرات في MyRibbon تشير إلى ActionsPaneControls الذين أضفناهما سابقا

- أضف الكود التالي لإجراء معالجة الحدث Load الخاص بـ MyRibbon الذي سيضيف التحكمات ActionsPane1 و ActionsPane2 إلى مجموعة ActionsPane ثم إخفاءهما

```
Private Sub MyRibbon_Load(ByVal sender As System.Object, _
    ByVal e As RibbonUIEventArgs) Handles MyBase.Load

      ' Add ActionsPane1 & ActionsPane2 to ActionPane Collection
      Globals.ThisWorkbook.ActionsPane.Controls.Add(ActionsPane1)
      Globals.ThisWorkbook.ActionsPane.Controls.Add(ActionsPane2)
      ' Hide Action Panes From View
      ActionsPane1.Hide()
      ActionsPane2.Hide()
      Globals.ThisWorkbook.Application.DisplayDocumentActionTaskPane = False
End Sub
```

- سنضيف الآن إجراء معالجة الحدث Click للزر الأول الذي سيقوم بإظهار ActionsPane1 وإخفاء ActionsPane2 ويكون الكود كالتالي

```
' Show Actions Pane 1 Button Event Handler
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As _
    Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) Handles Button2.Click

    Globals.ThisWorkbook.Application.DisplayDocumentActionTaskPane = True
    ActionsPane1.Show()
    ActionsPane2.Hide()
```

```
End Sub
```

- سنضيف الآن إجراء معالجة الحدث Click للزر الثاني الذي سيقوم بإظهار ActionsPane2 وإخفاء ActionsPane1 ويكون الكود كالتالي

```
' Show Actions Pane 2 Button Event Handler
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As _
    Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) Handles Button3.Click

    Globals.ThisWorkbook.Application.DisplayDocumentActionTaskPane = True
    ActionsPane2.Show()
    ActionsPane1.Hide()
End Sub
```

- سنضيف الآن إجراء معالجة الحدث Click لـ ToggleButton1 كما يلي

```
' Hide Actions Pane Event Handler
Private Sub ToggleButton1_Click(ByVal sender As System.Object, ByVal e As _
    Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) Handles ToggleButton1.Click

    If ToggleButton1.Checked Then
        Globals.ThisWorkbook.Application.DisplayDocumentActionTaskPane = False
    Else
        Globals.ThisWorkbook.Application.DisplayDocumentActionTaskPane = True
    End If
End Sub
```

- شغل التطبيق وانتقل إلى صفحة Add Ins لاختبار البرنامج هنا وإن قمت بإضافة صفحات إضافية ستراها في آخر صفحات شريط أدوات Excel 2007

أتمتة وورد 2007 باستخدام فيجول بايزيك دوت نيت

حتى نستطيع العمل نحتاج لبعض الخطوات الابتدائية

من خصائص المشروع ومن صفحة References اضغط الزر ADD ثم انتقل لصفحة Com وأضف مرجع للمكتبة Microsoft Word 12.0 Object Library ثم من أسفل الصفحة ومن القائمة أسفل Imported References انتقل إلى Microsoft.Office.Interop وضع إشارة اختيار بجانبها حتى تكون متوفرة للمشروع بأكمله وإلا ستضطر لاستخدام Imports التالية في بداية كل ملف ستستخدم أتمتة وورد فيه

```
Imports Microsoft.Office.Interop
```

و إن كنت ستستخدم أغراض Object لوثائق أو التطبيق الخاصة بوورد في عدة اجرائيات يفضل تعريفها كعناصر في النموذج الحاوي لتلك الإجرائيات - متغيرات عامة على مستوى النموذج - حتى لا تفقد اتصالك مع وورد

بدء وإنهاء وورد

عرف متغير عام على مستوى النموذج كما يلي

```
Private axWord As Word.Application
```

لبدء وورد نستخدم الكود التالي

```
axWord = New Word.Application
axWord.Visible = True
```

و لإنهاء وورد نستخدم الكود التالي

```
axWord.Quit()
```

إنشاء وفتح الوثائق

لإنشاء وثيقة جديدة ابدأ وورد أولا ثم استخدم الأمر Application.Documents.Add

```
Dim axDoc As Word.Document
axDoc = axWord.Documents.Add
```

لفتح وثيقة موجودة

```
Dim axDoc As Word.Document
axDoc = axWord.Documents.Open("c:\MyDocument.docx")
```

لحفظ الوثيقة

```
axDoc.SaveAs("C:\MyDocument.docx")
```

قراءة وإدراج النصوص

عندما تقوم بأتمتة وورد ستجد العديد من الطرائق التي تقوم بنفس الوظيفة وعملية قراءة وإدراج النصوص مثال جيد على ذلك حيث يمكنك عمل ذلك باستخدام العديد من الأشياء Objects ولكن أكثرها استخداما هو Paragraph و Selection

paragraph object

تتكون وثيقة وورد من مجموعة من الفقرات وهي مرتبة بالتسلسل بدءا من 1 وهي تتضمن كامل النص حتى محارف الإرجاع ويمكن الحصول على نص الفقرة الأولى

```
Dim strText As String
strText = axDoc.Paragraphs(1).Range.Text
```

ولتغيير نص الفقرة نستخدم

```
axDoc.Paragraphs(1).Range.Text = "Hello from Visual Basic 2005"
```

Selection object

وهو الطريقة الأكثر مرونة لإدراج النصوص وهو شبيه باستخدامك لوورد عند كتابتك لوثيقة فتستخدم أولا Selection object لنقل نقطة الإدراج للمكان المطلوب ضمن الوثيقة ثم تقوم بإدراج النص المطلوب والكود التالي يقوم بإدراج نص في آخر الوثيقة وفي بدايتها ثم يبحث عن كلمة ويدخل نص مباشرة بعدها

```
'Activate the document first
axDoc.Activate()
'Move to the end and add text
axWord.Selection.EndKey(Word.WdUnits.wdStory)
axWord.Selection.TypeText("This is the end")
'Move to the beginning and add text
axWord.Selection.HomeKey(Word.WdUnits.wdStory)
axWord.Selection.TypeText("This is the beginning")
axWord.Selection.Find.ClearFormatting()
'Locate Foo, then add text following it
axWord.Selection.Find.ClearFormatting()
axWord.Selection.Find.Text = "Foo"
axWord.Selection.Find.Execute()
axWord.Selection.MoveRight(Word.WdUnits.wdCharacter, 1)
axWord.Selection.TypeText("This is Foo")
```

إدراج الجداول

الكود التالي يبين طريقة إدراج جدول بقياس 5 × 5 و إضافة تنسيق للجدول ثم ضبط نص في خلية محددة في ذلك الجدول

```
Dim axTable As Word.Table
axTable = axDoc.Tables.Add(axWord.Selection.Range, 5, 5)
axTable.Style = "Table Grid 8"
axTable.Cell(3, 3).Range.Text = "Hello World"
```

الطباعة

لطباعة لوثيقة نستخدم الطريقة PrintOut

```
axDoc.PrintOut()
```

التأكد من أن الوثيقة مفتوحة

بما أن المستخدم يمكنه إغلاق الوثيقة أو الوورد في أي وقت رغم أن برنامجك مازال مرتبطا به يمكنك استخدام الطريقة التالية للتأكد من أن الوثيقة مازالت مفتوحة

```
Dim strName As String
Dim blnIsAvailable As Boolean
Try
    strName = axDoc.Name
    blnIsAvailable = True
Catch ex As Exception
```

```
    blnIsAvailable = False
End Try
MsgBox("Document is available: " & blnIsAvailable)
```

إدراج جدول من ADO .net

يمكنك استخدام الطريقة التالية لإدراج بيانات من جدول في قاعدة بيانات في جدول في وثيقة وورد

```
 Function AddDataTable(ByVal tbl As DataTable) As Boolean
     Dim nRowCount, nColCount, nRow, nCol As Integer
     Dim axTable As Word.Table
     nRowCount = tbl.Rows.Count
     nColCount = tbl.Columns.Count
     axWord.Selection.EndKey(wdStory)
     axTable = axDoc.Tables.Add(axWord.Selection.Range, nRowCount + 1, nColCount)
     axTable.Style = "Table Grid 8"
     For nCol = 1 To nColCount
         axTable.Cell(1, nCol).Range.Text = tbl.Columns.Item(nCol - 1).Caption
     Next
     For nCol = 1 To nColCount
         For nRow = 1 To nRowCount
             axTable.Cell(nRow + 1, nCol).Range.Text = tbl.Rows(nRow - 1).Item(nCol - 1)
         Next
     Next
     Return True
 End Function

' To use this function
tbl = Me.CustomersDataSet.Tables(0)
WordDoc.AddDataTable(tbl)
```

تحكمات المحتويات content controls

يقدم وورد 2007 آلية جديدة لإدراج البيانات في الوثائق تدعى تحكمات المحتويات Content Controls وهي عبارة عن حقول يمكن تحريرها يدويا أو ملؤها برمجيا أو نشرها من وثيقة XML وهي تتضمن عدة تحكمات مثل صندوق النصوص أو الصور أو القائمة المركبة ... الخ ويمكن الوصول إليها من خلال ContentControls collection في Document objects كما في المثال

```
Dim ccCollection As Word.ContentControls
ccCollection = axDoc.ContentControls
ccCollection.Item(1).Range.Text = " Content Control Field"
```

دمج البريد

لتحقيق العملية

افتح الوثيقة المصدر ثم اضبط الخاصية Mail Merge DataSource ثم يتم إنشاء وثيقة جديدة

وفيما يلي مثال عن كيفية عمل ذلك دون تدخل من المستخدم

```
axDoc.Activate()
axDoc.MailMerge.MainDocumentType = _
Word.WdMailMergeMainDocType.wdFormLetters
axDoc.MailMerge.OpenDataSource(strDatabaseFilename, _
wdOpenFormatAuto, False, False, False, False, "", "", _
False, "", "", strConnectionString, strSQL, "", False,_
Word.WdMergeSubType.wdMergeSubTypeOAL)
axDoc.MailMerge.DataSource.FirstRecord = wdDefaultFirstRecord
axDoc.MailMerge.DataSource.LastRecord = wdDefaultLastRecord
axDoc.MailMerge.Destination = _
Word.WdMailMergeDestination.wdSendToNewDocument
axDoc.MailMerge.Execute(False)
```

كتابة شيفرة لإنشاء Add-in يتم استدعاؤه من VBA

أنشئ مشروعا جديدا بلغة Visual Basic من نوع Office ثم 2007 ثم اختر Add-in 2007 Excel وقم بتسمية المشروع ExcelImportData حيث تكمن فكرة مشروعنا هنا في إنشاء فئة وكشفها لحلول أوفيس بحيث يمكننا استدعاء طرائق تلك الفئة من كود VBA حيث سيتم إنشاء فئة باسم AddInUtilities تمتلك طريقة تدعى ImportData

أضف فئة جديدة للمشروع باسم AddInUtilities وأضف الاستيرادات التالية لبداية الملف

```
Imports System.Data
Imports System.Runtime.InteropServices
Imports Excel = Microsoft.Office.Interop.Excel
```

استبدل جسم الفئة AddInUtilities بالكود التالي

```
<System.Runtime.InteropServices.ComVisibleAttribute(True)> _
<System.Runtime.InteropServices.InterfaceType(ComInterfaceType.InterfaceIsIDispatch)> _
Public Interface IAddInUtilities
    Sub ImportData()
End Interface

<System.Runtime.InteropServices.ComVisibleAttribute(True)> _
<System.Runtime.InteropServices.ClassInterface(System.Runtime.InteropServices.ClassInterf
aceType.None)> _
Public Class AddInUtilities
    Implements IAddInUtilities

    Public Sub ImportData() Implements IAddInUtilities.ImportData

        ' Create a new DataTable.
        Dim ds As New DataSet()
        Dim dt As DataTable = ds.Tables.Add("Customers")
        dt.Columns.Add(New DataColumn("LastName"))
        dt.Columns.Add(New DataColumn("FirstName"))

        ' Add a new row to the DataTable.
        Dim dr As DataRow = dt.NewRow()
        dr("LastName") = "Chan"
        dr("FirstName") = "Gareth"
        dt.Rows.Add(dr)

        ' Add a new XML map to the collection.
        Dim activeWorkbook As Excel.Workbook = _
               Globals.ThisAddIn.Application.ActiveWorkbook
        Dim xmlMap1 As Excel.XmlMap = activeWorkbook.XmlMaps.Add(ds.GetXmlSchema(), _
            "NewDataSet")

        ' Import the data.
        If Not (xmlMap1 Is Nothing) Then
            Dim lastSheet As Object = activeWorkbook.Sheets(activeWorkbook.Sheets.Count)
            Dim newSheet As Excel.Worksheet = CType(activeWorkbook.Sheets.Add( _
                After:=lastSheet), Excel.Worksheet)
            newSheet.Name = "Imported Data"
            activeWorkbook.XmlImportXml(ds.GetXml(), xmlMap1, True, _
                newSheet.Range("A1"))
        End If
    End Sub
End Class
```

حيث تقوم بتعريف واجهة في بداية الكود تدعي IAddInUtilities تمتلك الخاصية ComVisible مضبوطة إلى True والخاصية InterfaceType تمتلك القيمة IntetrfaceIsDispatch ويكمن السبب في كتابة هذه الواجهة هو أن الفئة الأساسية سوف تحقق وتعرض الواجهة IDispatch مما يجعلها متوفرة للاستدعاء من قبل حلول أوفيس الأخرى

```
<System.Runtime.InteropServices.ComVisibleAttribute(True)> _
<System.Runtime.InteropServices.InterfaceType(ComInterfaceType.InterfaceIsIDispatch)> _
Public Interface IAddInUtilities
    Sub ImportData()
End Interface
```

وتعرف الفئة AddInUtilities لتحقق الواجهة IAddInUtilities وتطبق الخاصية ComVisisble على هذه الفئة

```
<System.Runtime.InteropServices.ComVisibleAttribute(True)> _
<System.Runtime.InteropServices.ClassInterface(System.Runtime.InteropServices.ClassInterf
aceType.None)> _
Public Class AddInUtilities
    Implements IAddInUtilities
```

ثم تعرف الطريقة ImportData والتي تحقق الطريقة ImportData المحددة في الواجهة IAddInUtilities في بداية الكود فتنشئ أولا Dataset مع DataTable ثم يضاف سطر جديد لجدول البيانات ثم يضاف Xml Map مبنية على Dataset to the Collection of XML maps ثم تضاف صفحة جديدة للملف ثم تستخدم Xml Map لاستيراد البيانات من الـ Dataset إلى صفحة الـ Excel.

وبهذا تكون فئتنا قد انتهت ولكنها مازالت غير متوفرة لبقية حلول الأوفيس. الآن افتح محرر الكود للفئة ThisAddIn وقم بتجاوز Override الطريقة RequestComAddInAutomationService معيدا منها مرجعا للفئة AddInUtilities كما في الكود

```
Private utilities As AddInUtilities
 Protected Overrides Function RequestComAddInAutomationService() As Object
     If utilities Is Nothing Then
         utilities = New AddInUtilities()
     End If
     Return utilities
 End Function
```

وللاختبار قم بتشغيل المشروع ثم قم بحفظه كـ Excel Macro-Enabled Workbook (*.xlsm) ثم انتقل إلى صفحة developers ثم انقر Visual Basic ليفتح لك محرر الكود وافتح ThisWorkBook وأضف الكود التالي

```
Sub CallVSTOMethod()
    Dim addIn As COMAddIn
    Dim automationbject As Object
    addIn = Application.COMAddIns("ExcelImportData")
    automationObject = addIn.Object
    automationObject.ImportData()
End Sub
```

الذي يعرف متغيرا يشير للغرض ComAddIn الذي يمثل الـ ExcelImportData Add-in ثم يستخدم لاستدعاء الطريقة ImportData من فئتنا السابقة.

الآن عد إلى ملف الـ Excel ثم شغل الماكرو CallVSTOMethod الذي قمنا بإنشائه للتو وربما ستظهر لك رسالة تفيد بأنه لا توجد Schema مرتبطة ببيانات الـ XML وأن الـ Excel سيقوم بإنشائها لك فقط قم بالموافقة على الرسالة هنا فتلاحظ إنشاء ورقة جديدة باسم Imported Data وقد تم إضافة بيانات في الخلية A1 والخلية B1

كيف نستدعي صناديق الحوار الخاصة بمايكروسوفت وورد من برنامجنا

عندما تتعامل مع مايكروسوفت وورد تأتيك أوقات تحتاج لإظهار صناديق حوار للحصول على دخل من المستخدم. ورغم وجود إمكانية لإنشاء صناديق الحوار الخاصة بك إلا أنه يمكنك أيضا الاستفادة من صناديق الحوار الموجودة سلفا والتي يمكننا الوصول إليها من خلال المجموعة Dialogs في الغرض Application وهي أكثر من 200 صندوق حوار تم تقديمها على شكل تعدادات. فإذا أدرنا استدعاء صندوق حوار إنشاء مستند جديد على سبيل المثال يمكننا عمل ذلك بتمرير القيمة Word.WdWordDialog.wdDialogFileNew للدالة Application.Dialogs.Item كما في الكود التالي وللتجربة استخدم مشروعا من النوع Word 2007 Document

```
Dim dlg As Word.Dialog = _
        Application.Dialogs.Item(Word.WdWordDialog.wdDialogFileNew)
dlg.Show()
```

وللوصول إلى عناصر صندوق الحوار نستخدم الطريقة InvokeMember لتحديد اسم الملف لصندوق حوار فتح ملف بالطريقة الظاهرة بالكود التالي حيث نحصل على النوع Type الخاص بصندوق الحوار الذي نستخدمه ثم نضبط قيمة الخاصية المطلوبة باستخدام الطريقة InvokeMember

```
Dim dlg As Word.Dialog = Application.Dialogs(Word.WdWordDialog.wdDialogFileOpen)
    Dim dlgType As Type = GetType(Word.Dialog)

    ' Set the Name property of the dialog box.
    dlgType.InvokeMember("Name", _
        Reflection.BindingFlags.SetProperty Or _
            Reflection.BindingFlags.Public Or _
            Reflection.BindingFlags.Instance, _
        Nothing, dlg, New Object() {"Testing"}, _
        System.Globalization.CultureInfo.InvariantCulture)

    dlg.Show()
```

كما يمكننا الحصول على قيمة أي خاصية لصندوق حوار خاص بمايكروسوفت وورد بنفس الطريقة تقريبا وذلك بتمرير القيمة GetProperty للطريقة InvokeMember بدلا عن القيمة SetProperty كما يرينا ذلك المثال التالي

```
' Show the Name property.
MessageBox.Show(dlgType.InvokeMember("Name", _
    Reflection.BindingFlags.GetProperty Or _
    Reflection.BindingFlags.Public Or _
    Reflection.BindingFlags.Instance, _
    Nothing, dlg, Nothing, _
System.Globalization.CultureInfo.InvariantCulture))
```

ويمكننا القيام ببعض الإجراءات المعقدة باستخدام صناديق الحوار المضمنة سلفا مع مايكروسوفت وورد باستخدام هذه الصناديق بدون إظهارها للمستخدم وذلك باستدعاء الطريقة Execute لصندوق الحوار بدون استدعاء الطريقة Display كما في المثال التالي الذي يتطلب استخدام الضبط Option Strict Off بسبب أن قيم الخصائص التي نتعامل معها هنا ليست عناصر ضمن الفئة Dialog أساسا وهي تستخدم الربط المتأخر حيث يتم إنشاؤها ديناميكيا في زمن التشغيل عندما يقوم المترجم بتقييم التعداد wdDialogFilePageSetup حيث يتم إنشاء هذه الخصائص لتكون مطابقة للتحكمات التي ستظهر على صندوق الحوار ومثالنا هنا يستخدم التعداد wdDialogFilePageSetup لضبط عدة خصائص للصفحة بدون طلب الإدخال من المستخدم حيث يستخدم الكود الغرض Dialog لضبط حجم مخصص للصفحة

```
Friend Sub PageSetupDialogHidden()
    Dim dlg As Word.Dialog = _
            Application.Dialogs.Item(Word.WdWordDialog.wdDialogFilePageSetup)

    ' Set the properties of the dialog box.
    ' ControlChars.Quote() is used to represent the symbol for inches.
    With dlg
```

```
        .PageWidth = 3.3 & ControlChars.Quote
        .PageHeight = 6 & ControlChars.Quote
        .TopMargin = 0.71 & ControlChars.Quote
        .BottomMargin = 0.81 & ControlChars.Quote
        .LeftMargin = 0.66 & ControlChars.Quote
        .RightMargin = 0.66 & ControlChars.Quote
        .HeaderDistance = 0.28 & ControlChars.Quote
        .Orientation = Word.WdOrientation.wdOrientPortrait
        .DifferentFirstPage = False
        .FirstPage = 0
        .OtherPages = 0

        ' Apply these settings only to the current selection
        ' with this line of code:
        .ApplyPropsTo = 3

        ' Apply the settings.
        .Execute()
    End With
End Sub
```

القسم الثالث - Silverlight و WPF و XAML

ويضم المواضيع التالية:

- كتابة تطبيقك الأول من النوع WPF Application
- كيف نطبق مظهر مختلف على التحكمات في تطبيق WPF
- كيف يمكننا تضمين صورة كمصدر في تطبيق wpf ثم إظهارها وقت التنفيذ
- كيف يمكننا تطبيق مظهر مخصص لنافذة برنامج WPF في زمن التشغيل
- استخدام تحكمات Windows Forms من داخل تطبيق WPF
- استخدام عناصر WPF من داخل تطبيق Windows Forms
- كتابة تطبيقنا الأول بتقنية Silverlight
- أدوات التحكم بترتيب العناصر Silverlight and WPF
- إنشاء ساعة تماثلية باستخدام تقنية SilverLight باستخدام الكود
- الفروقات في معالجة xaml بين Silverlight و WPF
- كيف نستخدم عناصر Style للتحكم بمظهر التطبيق
- تخصيص مظهر التحكمات Silverlight

كتابة تطبيقك الأول من النوع WPF Application

تصميم تطبيق WPF لا يختلف كثيرا عن تصميم تطبيق Windows Forms حيث يمكنك إضافة التحكمات على سطح التصميم ومع ذلك فهناك بعض الاختلافات فبالإضافة لنوافذ التصميم المألوفة ستجد نافذة إضافية تعرض كود XAML حيث تستخدم هذه اللغة لإنشاء واجهة المستخدم. فعندما تقوم بتطوير تطبيقات النوافذ التقليدية تقوم بسحب التحكمات من الـ ToolBox أو إن رغبت يمكنك كتابة كود لإنشاء تلك التحكمات فعندما تقوم بسحب التحكم إلى النموذج يتم إنشاء الكود آليا من أجلك وبشكل مشابه عندما تقوم بإنشاء تطبيق WPF يمكنك إنشاء التحكم بكتابة كود XAML أو بسحب التحكم إلى نافذة WPF.
ولغة XAML منظمة إلى عناصر ببنية شجرية مشابهة لطريقة كتابة ملفات XML حيث تجد أن العنصر مضمن ضمن قوسين حادين وهناك تحديد فتح وتحديد إغلاق لكل عنصر فمثلا قد يكون لديك عنصر Button يمكن أن يظهر بالشكل <Button></Button> ويمكنك أن تصف كيف سيبدو ذلك العنصر بضبط الخصائص attributes كـ Location أو Height أو Width وهي تظهر داخل الأقواس الافتتاحية للعنصر بشكل زوج اسم وقيمة مفصولين بعلامة = حيث تكون القيمة محصورة ضمن علامتي تنصيص مثل

```
<Button Height="23"></Button>
```

وعندما تقوم بسحب تحكمات WPF من الـ ToolBox إلى نافذة المصمم تولد بيئة التطوير كود XAML آليا لذلك التحكم فالكود التالي يتم توليده نتيجة عملية إضافة تحكم من النوع System.Windows.Controls.Button

```
<Button Height="23" HorizontalAlignment="Left" Margin="10,10,0,0"
    Name="Button1" VerticalAlignment="Top" Width="75">Button</Button>
```

دعنا نقم ببعض التجارب تطبيقا على ما ورد سابقا:

- أنشئ تطبيقا جديدا من النوع WPF Application و سمه WPFWindow ثم اضغط OK فيتم إنشاء تطبيق WPF جديد من أجلك وسترى قسم XAML جديد تطبيقات WPF وستبدو محتوياته كالتالي
-

```
<Window x:Class="Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>

    </Grid>
</Window>
```

- انقر على Window1 لاختيارها وفي نافذة محرر XAML غير قيمة الخاصية Title إلى WPF Application فنلاحظ تغير نص عنوان النافذة إلى WPF Application كما يمكنك تجريب تغيير قيمة بعض الخصائص الأخرى
- قم بسحب TextBox من الـ ToolBox وضعه في الزاوية اليمينية العليا لنافذة التطبيق ثم انقر عليه لاختياره وفي نافذة الخصائص اضبط قيمة HorizontalAlignment إلى Left ثم قم بضبط الخاصية VerticalAlignment إلى Top والخاصية Width إلى 75 والخاصية Height إلى 26
- في محرر XAML قم بتغيير الخاصية Width إلى 140 والعنصر Margin إلى 30, 56, 0, 0 كما يبدو في المثال

```
<TextBox Height="26" HorizontalAlignment="Left" Margin="30,56,0,0"
    Name="TextBox1" VerticalAlignment="Top" Width="140" />
```

- قم بسحب تحكم Button إلى سطح نافذة التطبيق وقم بالنقر المزدوج عليه لإنشاء إجراء معالجة الحدث Click الخاص به ثم اكتب الكود التالي في الإجراء

```
MsgBox("Event handler was created by double-clicking the button.")
```

- أضف تحكم Button آخر للنافذة وقم باختياره ثم قم بإضافة خاصية باسم Click للعنصر Button في محرر كود XAML واضبط قيمتها إلى ButtonOKClicked وهذا الاسم سوف تعطيه لإجراء معالجة الحدث في الكود مع ملاحظة أنه عندما تنشئ

إجراء معالجة الحدث بالنقر المزدوج لا يتم إضافة الخاصية Click إلى كود XAML ولكن يستخدم في الكود عبارة Handles عوضا عن ذلك

- انقر بزر الفأرة اليميني على نافذة التصميم واختر ViewCode ثم قم بإضافة الإجراء التالي

```
Sub ButtonOKClicked(ByVal Sender As Object, _
    ByVal e As RoutedEventArgs)

    MsgBox("Event handler was created manually.")

End Sub
```

- اضغط F5 لتنفيذ المشروع واختباره

عمل برنامج رسم باستخدام WPF

يمكننا عمل برنامج رسم بسيط تطبيقا على بعض ما ورد أعلاه

- أنشئ مشروعا جديدا من النوع WPF Application وسمه Ink Pad
- اضبط حجم نافذة التطبيق إلى 550 ارتفاعا و 370 عرضا وذلك بضبط قيم الخصائص Height و Width على التوالي
- غير عنوان نافذة WPF إلى Ink Pad وذلك بضبط الخاصية Title
- غير الخاصية Background لنافذة WPF إلى Brown كما يمكننا ضبط قيمة الخاصية في محرر XAML بإضافة الخاصية Background لكود النافذة فيصبح كود XAML للنافذة كما يلي

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="550" Width="370" Background="Brown">
    <Grid>

    </Grid>
</Window>
```

- افتح صندوق الأدوات ToolBox ثم انقر عليه بزر الفأرة اليميني واختر Choose Items من القائمة ثم انتقل لصفحة WPF Components في صندوق حوار Choose ToolBox Items وانتقل عبر القائمة إلى InkCanvas وقم باختيارها – تأكد من وضع علامة في المربع – ثم اضغط OK
- اسحب InkCanvas من صندوق الأدوات ToolBox إلى سطح النافذة واضبط القيم التالية:
  - o الخاصية Width و الخاصية Height إلى Auto
  - o الخاصية HorizontalAlignment و الخاصية Vertical Alignment إلى Stretch
  - o الخاصية Margins إلى 9,9,9,68
  - o الخاصية Background إلى LightYellow وسترى أن لون الخلفية أصبح أصفر عندما تشغل البرنامج
- اسحب زرين من صندوق الأدوات إلى سطح النافذة أسفل Ink Canvas وضع Button1 إلى اليسار و Button2 إلى اليمين

• اختر Button1 واضبط كود XAML الخاص به إلى

```
<Button Height="23" HorizontalAlignment="Left" Margin="85,0,0,24"
Name="Button1" VerticalAlignment="Bottom" Width="75">Clear</Button>
```

• اختر Button2 واضبط كود XAML الخاص به إلى

```
<Button Height="23" HorizontalAlignment="Right" Margin="0,0,72,24"
Name="Button2" VerticalAlignment="Bottom" Width="75">Close</Button>
```

• وبهذا يصبح كود XAML لنافذة WPF كما يلي

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="550" Width="370" Background="Brown">
    <Grid>
        <InkCanvas Height="Auto"  Margin="9,9,9,68" Name="InkCanvas1"
VerticalAlignment="Stretch" Width="Auto" HorizontalAlignment="Stretch"
Background="LightYellow" />

        <Button Height="23" HorizontalAlignment="Left" Margin="85,0,0,24"
            Name="Button1" VerticalAlignment="Bottom" Width="75">Clear</Button>

        <Button Height="23" HorizontalAlignment="Right" Margin="0,0,72,24"
            Name="Button2" VerticalAlignment="Bottom" Width="75">Close</Button>

    </Grid>
</Window>
```

• انقر نقرا مزدوجا على الزر Clear ثم أضف الكود التالي في إجراء معالجة الحدث Click للزر

```
Me.InkCanvas1.Strokes.Clear()
```

• وبنفس الطريقة أضف الكود التالي في إجراء معالجة الحدث Click للزر Close

```
Me.Close
```

• شغل البرنامج واختبره

كيف نطبق مظهر مختلف على التحكمات في تطبيق WPF

- أنشئ تطبيق WPF جديد ثم أضف الكود `Background="Azure"` في نهاية السطر الذي يبدأ بـ Title وقبل القوس > لتغيير لون خلفية النموذج إلى الأزرق الفاتح وسنقوم بتغيير مظهر الأزرار في مقالنا هذا لنعطي فكرة عن كيفية القيام بذلك
- استبدل <Grid> بـ <StackPanel> و </Grid> بـ </StackPanel> في محرر Xaml الخاص بـ Window1
- أضف الكود التالي بعد <StackPanel> مباشرة لكي نقوم بإضافة ComboBox للنافذة

```
<ComboBox Width="100px" Height="Auto">
    <ComboBoxItem>Item One</ComboBoxItem>
    <ComboBoxItem>Item Tow</ComboBoxItem>
    <ComboBoxItem>Item Three</ComboBoxItem>
</ComboBox>
```

- أضف الكود التالي بعد الكود السابق مباشرة كي نضيف StackPanel آخر للنافذة مع تحديد أن الاصطفاف سيكون نحو المركز والاتجاه أفقي

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">

</StackPanel>
```

- داخل الـ StackPanel الذي أضفناه مؤخرا أضف الكود التالي الذي يقوم بإضافة ثلاثة أزرار Add و Edit و Delete للنموذج

```
<Button>Add</Button>
<Button>Edit</Button>
<Button>Delete</Button>
```

- حتى هذه النقطة يجب أن يكون كود xaml الخاص بالنافذة لديك على الشكل

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300" Background="Azure">
    <StackPanel>
        <ComboBox Width="100px" Height="Auto">
            <ComboBoxItem>Item One</ComboBoxItem>
            <ComboBoxItem>Item Tow</ComboBoxItem>
            <ComboBoxItem>Item Three</ComboBoxItem>
        </ComboBox>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button>Add</Button>
            <Button>Edit</Button>
            <Button>Delete</Button>
        </StackPanel>
    </StackPanel>
</Window>
```

- سنقوم الآن بتطبيق بعض الخصائص التي ستغير مظهر الأزرار استبدل الكود الخاص بالزر Delete بالكود التالي

```
<Button Margin="10,5,2,5">
    <Button.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <GradientStop Offset="0.25" Color="#FFFF1122"></GradientStop>
            <GradientStop Offset="0.85" Color="#FFFFFFFF"></GradientStop>
        </LinearGradientBrush>
    </Button.Background>
 Delete</Button>
```

- حيث استخدمنا الخاصية Margin لتحديد الهوامش الخاصة بالزر ثم استخدمنا الفرشاة LinearGradientBrush لملئ الزر بلون متدرج حيث حددت الخاصية StartPoint نقطة بدء الملئ و EndPoint نقطة النهاية وحدد قسم GradientStop الأول والثاني اللونين الخاصين بالتدرج والإزاحة الخاصة بكل لون حيث نلاحظ بعد تطبيق هذه الخاصة تغير مظهر الزر Delete إلى لون متدرج من الأحمر للأبيض
- وإن أردنا تطبيق مظهر واحد لجميع الأزرار في النموذج نضيف قسم <Window.Resources> في أعلى نافذة كود xaml وبعد الأسطر العلوية التي تحدد النموذج حيث نفتح قسم Style بداخله ونحدد أن النوع المستهدف من هذا المظهر هو Button

```
<Window.Resources>
    <Style TargetType="Button">


    </Style>
</Window.Resources>
```

- وبداخل قسم Style نفتح قسم آخر هو Setter ونحدد فيه ان الخاصية المستهدفة هي Background كما يلي

```
<Setter Property="Background">



</Setter>
```

- ثم نحدد المظهر كما فعلنا سابقا مع الزر Delete وذلك باستخدام LinearGradientBrush وبنفس الطريقة السابقة تماما فيصبح كود قسم setter كما يلي

```
<Setter Property="Background">
    <Setter.Value>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <GradientStop Offset="0.3" Color="#FFFFFF55"/>
            <GradientStop Offset="0.8" Color="#AA5555FF"/>
        </LinearGradientBrush>
    </Setter.Value>
</Setter>
```

- ومباشرة بعد قسم setter الذي أضفناه للتو سنضيف قسم Setter آخر الغرض منه هو تحديد الهوامش الخاصة بالأزرار وسيكون الكود الخاص به كما يلي

```
<Setter Property="Margin" Value="2,5,2,5"></Setter>
```

- وبذلك يصبح كود xaml الكامل للنافذة حتى هذه النقطة كما يلي

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300" Background="Azure">

    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.3" Color="#FFFFFF55"/>
                        <GradientStop Offset="0.8" Color="#AA5555FF"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="Margin" Value="2,5,2,5"></Setter>
        </Style>
    </Window.Resources>

    <StackPanel>
        <ComboBox Width="100px" Height="Auto">
```

```
            <ComboBoxItem>Item One</ComboBoxItem>
            <ComboBoxItem>Item Tow</ComboBoxItem>
            <ComboBoxItem>Item Three</ComboBoxItem>
        </ComboBox>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button>Add</Button>
            <Button>Edit</Button>
            <Button Margin="10,5,2,5">
                <Button.Background>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.25" Color="#FFFF1122"></GradientStop>
                        <GradientStop Offset="0.85" Color="#FFFFFFFF"></GradientStop>
                    </LinearGradientBrush>
                </Button.Background>
                Delete
            </Button>
        </StackPanel>
    </StackPanel>
</Window>
```

- حيث نلاحظ أن المظهر الذي قمنا بتحديده كمظهر عام لجميع الأزرار في النموذج قد تم تطبيقه على الزرين Add و Edit ولم يتم تطبيقه على الزر Delete وذلك بسبب أن الزرين Add و Delete لم نقم بتحديد مظهر خاص بهما فيأخذان المظهر العام الذي قمنا بتحديده في البداية لجميع الأزرار في النموذج وبما أننا قمنا بتحديد مظهر خاص بالزر Delete ملتصق به مباشرة لذا يأخذ المظهر الأقرب له
- أضف زرا جديدا وليكن Search ضمن مجموعة أزرار النموذج وقبل الزر add مباشرة وسنقوم الآن بإضافة مظهر مخصص له بطريقة أخرى وهي تحديد Key يميز هذا المظهر حيث نربطه مع الزر الذي نرغب بتطبيقه عليه باستخدام الخاصية Style لذلك الزر ولعمل ذلك قم بإضافة كود قسم style التالي بعد قسم Style السابق مباشرة وقبل </Window.Resources>

```
<Style TargetType="Button" x:Key="MySearch">
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                <GradientStop Offset="0.3" Color="#FF11FF11"/>
                <GradientStop Offset="0.8" Color="#CCFFBB11"/>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="Margin" Value="2,5,10,5"></Setter>
</Style>
```

- حيث نلاحظ إضافة الخاصية x:Key في بداية تعريف قسم Style والتي يمكننا من خلالها ربط أي زر مع هذه الخاصية حيث سنقوم بتعديل كود الزر Search ليرتبط مع المظهر الجديد ليصبح كما يلي

```
<Button Style="{StaticResource MySearch}">Search</Button>
```

وبهذا نكون قد غطينا فكرة مبدئية عن كيفية تغيير مظهر التحكمات في برنامج WPF برمجيا وفيما يلي سرد كامل لكود Xaml الخاص بهذا الموضوع متضمن كافة الأفكار التي تم طرحها هنا

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300" Background="Azure">

    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.3" Color="#FFFFFF55"/>
                        <GradientStop Offset="0.8" Color="#AA5555FF"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
```

```
            <Setter Property="Margin" Value="2,5,2,5"></Setter>
        </Style>
        <Style TargetType="Button" x:Key="MySearch">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.3" Color="#FF11FF11"/>
                        <GradientStop Offset="0.8" Color="#CCFFBB11"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="Margin" Value="2,5,10,5"></Setter>
        </Style>
    </Window.Resources>

    <StackPanel>
        <ComboBox Width="100px" Height="Auto">
            <ComboBoxItem>Item One</ComboBoxItem>
            <ComboBoxItem>Item Tow</ComboBoxItem>
            <ComboBoxItem>Item Three</ComboBoxItem>
        </ComboBox>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Style="{StaticResource MySearch}">Search</Button>
            <Button>Add</Button>
            <Button>Edit</Button>
            <Button Margin="10,5,2,5">
                <Button.Background>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.25" Color="#FFFF1122"></GradientStop>
                        <GradientStop Offset="0.85" Color="#FFFFFFFF"></GradientStop>
                    </LinearGradientBrush>
                </Button.Background>
                Delete
            </Button>
        </StackPanel>
    </StackPanel>
</Window>
```

كيف يمكننا تضمين صورة كمصدر في تطبيق wpf ثم إظهارها وقت التنفيذ؟

- أنشئ مشروعا جديدا من النوع WPF Application
افتح مجلد المشروع وانسخ غليه الصورة التي تريدها
في Solution Explorer انقر بزر الفأرة اليميني على اسم مشروعك ثم اختر Add ثم existing Item لملفات المشروع
تأكد من أن خاصية Build Action موضوعة علىResource
أضف تحكم image وزر Button لنافذة المشروع
انقر نقرا مزدوجا على الزر لإنشاء إجراء لمعالجة الحدث Click للزر ثم أدخل الكود التالي وذلك بافتراض أن اسم ملف الصورة seatbelt.bmp

```
Me.Image1.Source = New BitmapImage(New
Uri("pack://application:,,,/seatbelt.bmp"))
```

- شغل المشروع واختبره
حيث استخدمنا الفئة URI للحصول على المصدر Resource والتي تعيد تمثيلا لغرض Object لمحدد مصادر موحد uniform Resource Identifierوتسهيل الوصول لأقسام URI

كيف يمكننا تطبيق مظهر مخصص لنافذة برنامج WPF في زمن التشغيل

أنشئ تطبيق WPF جديد

سنقوم بالبداية بإنشاء عمودين داخل الشبكة الخاصة بالنافذة بحيث يكون الثاني بضعف حجم الأول وذلك بإدخال الكود التالي بعد <Grid> مباشرة وذلك في محرر XAML الخاص بنافذة المشروع

```
<Grid.ColumnDefinitions>
     <ColumnDefinition Width="*"/>
     <ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>
```

كما سنقوم بقسم الشبكة إلى سطرين ولكن سنترك للبرنامج تحديد حجمهما ديناميكيا وذلك بإدخال الكود التالي بعد الكود السابق

```
<Grid.RowDefinitions>
      <RowDefinition/>
      <RowDefinition/>
</Grid.RowDefinitions>
```

سنضيف الآن بعض التحكمات للنموذج وسنبدأ بإضافة تحكم Label بحيث تكون محتوياته Street Address و التجانب الأفقي لليمين والشاقولي للوسط كما سنضيف صندوق نصوص يحتوي النص Enter Street Address وسيكون التجانب الأفقي لليسار والشاقولي للوسط وذلك بإدخال الكود التالي مباشرة بعد الكود السابق

```
<Label Content="Street Address" HorizontalAlignment="Right" VerticalAlignment="Center"></Label>
<TextBox Text="Enter a street address" HorizontalAlignment="Left"
VerticalAlignment="Center"></TextBox>
```

سنحدد الآن اسما لصندوق النصوص وليكن txtAddress وذلك بتحديد قيمة الخاصية x:Name إلى القيمة المطلوبة كما سنحدد مكانه في الشبكة أيضا بتحديد قيمة الخاصيتين Grid.Column و Grid.Row فيصبح الكود الخاص بصندوق النصوص على الشكل التالي بعد التعديل

```
<TextBox Grid.Column="1" Grid.Row="0" x:Name="txtAddress" Text="Enter a street address"
        HorizontalAlignment="Left" VerticalAlignment="Center"></TextBox>
```

كما سنقوم بتحديد مكان الـ Label في الشبكة باستخدام Grid.Column و Grid.row وذلك بنفس الطريقة السابقة إضافة إلى أننا سنقوم بربط الـ Lable بصندوق النصوص باستخدام الخاصية Target لـ Label فيصبح الكود الخاص بـ Label على الشكل التالي

```
<Label Grid.Column="0" Grid.Row="0" Target="{Binding ElementName=txtAddress,
Mode=OneWay}"
    Content="Street Address" HorizontalAlignment="Right"
VerticalAlignment="Center"></Label>
```

الآن سنقوم بإضافة زر أوامر باستخدام الكود التالي

```
<Button Grid.Column="1" Grid.Row="1" Content="Save Street Address"
        HorizontalAlignment="Left" VerticalAlignment="Top" />
```

أضف المحرف _ في بداية نص Contenet الخاص بـ Label لتصبح قيمة تلك الخاصية

```
Content="_Street Address"
```

وذلك حتى يمكننا استخدام الخاصية Target للانتقال مباشرة لصندوق النصوص باستخدام الاختصار Alt-S بما أننا وضعنا المحرف _ قبل الحرف S

شغل البرنامج واختبره ويكون قد أصبح كود xaml الخاص بالنافذة كما يلي

```
<Window x:Class="Window1"
```

```
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Test WPF Skinning" Height="300" Width="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Label Grid.Column="0" Grid.Row="0" Target="{Binding ElementName=txtAddress, Mode=OneWay}"
               Content="_Street Address" HorizontalAlignment="Right"
               VerticalAlignment="Center"></Label>
        <TextBox Grid.Column="1" Grid.Row="0" x:Name="txtAddress" Text="Enter a street address"
                 HorizontalAlignment="Left" VerticalAlignment="Center"></TextBox>
        <Button Grid.Column="1" Grid.Row="1" Content="Save Street Address"
                HorizontalAlignment="Left" VerticalAlignment="Top" />
    </Grid>
</Window>
```

انتقل إلى Solution Explorer وقم بفتح الملف Application.xaml كي نقوم بتحديد المظهر الخاص بأدواتنا هنا وسنبدأ بتحديد المظهر الخاص بـ Label وذلك بإدراج قسم Style التالي بعد <Application.Resources> مباشرة حيث سنقوم بتحديد قيمة اللون الأمامي وحجم الخط

```
<Style TargetType="{x:Type Label}" x:Key="LabelStyle">
    <Setter Property="Foreground" Value="Blue"></Setter>
    <Setter Property="FontSize" Value="12px"></Setter>
</Style>
```

وسنقوم الآن بتحديد مظهرين فارغين لكلا صندوق النصوص وزر الأوامر بحيث لن يغيرا شيئا في الوقت الحالي على مظهر التحكمين وذلك بإدخال الكود التالي مباشرة بعد الكود السابق

```
<Style TargetType="{x:Type TextBox}" x:Key="TextBoxStyle"></Style>
<Style TargetType="{x:Type Button}" x:Key="ButtonStyle"></Style>
```

وبهذا يصبح الكود الخاص بـ Application.xaml حتى الآن كالتالي

```
<Application x:Class="Application"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml">
    <Application.Resources>
        <Style TargetType="{x:Type Label}" x:Key="LabelStyle">
            <Setter Property="Foreground" Value="Blue"></Setter>
            <Setter Property="FontSize" Value="12px"></Setter>
        </Style>
        <Style TargetType="{x:Type TextBox}" x:Key="TextBoxStyle"></Style>
        <Style TargetType="{x:Type Button}" x:Key="ButtonStyle"></Style>
    </Application.Resources>
</Application>
```

ولربط هذه المظاهر الجديدة مع التحكمات نعود لكود xaml الخاص بالنافذة ونضيف الخاصية Style لكل من الـ Label و صندوق النصوص و زر الأوامر فيصبح الكود الخاص بالتحكمات الثلاثة على الشكل

```
<Label Grid.Column="0" Grid.Row="0" Target="{Binding ElementName=txtAddress, Mode=OneWay}"
       Content="_Street Address" HorizontalAlignment="Right" VerticalAlignment="Center"
               Style="{DynamicResource LabelStyle}"></Label>
<TextBox Grid.Column="1" Grid.Row="0" x:Name="txtAddress" Text="Enter a street address"
         HorizontalAlignment="Left" VerticalAlignment="Center"
         Style="{DynamicResource TextBoxStyle}"></TextBox>
<Button Grid.Column="1" Grid.Row="1" Content="Save Street Address"
        HorizontalAlignment="Left" VerticalAlignment="Top" Style="{DynamicResource ButtonStyle}"/>
```

ونلاحظ هنا أننا استخدمنا DynamicResource في الخاصية Style لربط مع الشكل المراد حيث نحدد أن هذه الخاصية يمكن أن يتم تغييرها في زمن التشغيل ديناميكيا وأن المظهر المطلوب قد يكون موجودا وقد لا يكون موجودا بينما لو استخدمنا StaticResource في هذه النقطة سنخبر البرنامج أننا نتوقع وجود ذلك المظهر وأنه لن يتم تغييره

شغل البرنامج واختبره

أضف الخاصية Click لتعريف الزر وذلك لإضافة إجراء لمعالجة حدث النقر على الزر بحيث تكون على الشكل

```
Click="Button_Click"
```

ويصبح تعريف الزر في النهاية على الشكل

```
<Button Grid.Column="1" Grid.Row="1" Content="Save Street Address" Click="Button_Click"
        HorizontalAlignment="Left" VerticalAlignment="Top" Style="{DynamicResource ButtonStyle}"/>
```

ثم انتقل لمحرر الكود الخاص بالملف Window1.xaml.vb وأضف إجراء معالجة حدث النقر على الزر وأضف استيرادا لـ System.IO و لـ System.Windows.Markup في بداية الملف وأنشئ إجراءا فارغا لحدث النقر على الزر كما يظهر في الكود التالي

```
Imports System.IO
Imports System.Windows.Markup

Class Window1
    Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)

    End Sub
End Class
```

اختر Add New Item من القائمة Project ثم أضف ملفا من النوع WPF Resource Dictionary للمشروع وقم بتسميته CrazyStyle.xaml وقم بتعديل الخاصية Copy to output directory الخاصة به إلى Copy if newer وأيضا قيمة الخاصية Build Action إلى Content حيث سنستخدمه كملف Skin للنافذة بحيث أننا سنستخدم الخاصية Dynamic Resources في WPF لتعمل كما لو أننا نستخدم Skin للمشروع وقم بتعديل محتويات كود xaml الخاص به ليصبح كما في الكود التالي حيث سيحتوي على الأشكال الخاصة بتحكمات نافذة المشروع وبنفس الطريقة التي استخدمناها منذ قليل

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style TargetType="{x:Type Label}" x:Key="LabelStyle">
        <Setter Property="Foreground" Value="Red" />
        <Setter Property="FontSize" Value="16px" />
    </Style>
    <Style TargetType="{x:Type TextBox}" x:Key="TextBoxStyle">
        <Setter Property="Background">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                    <GradientStop Offset=".1" Color="AliceBlue" />
                    <GradientStop Offset=".5" Color="AntiqueWhite" />
                    <GradientStop Offset="1" Color="Aquamarine" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Style>
    <Style TargetType="{x:Type Button}" x:Key="ButtonStyle">
        <Setter Property="Foreground" Value="Blue" />
        <Setter Property="FontSize" Value="16px" />
    </Style>

</ResourceDictionary>
```

الآن عد إلى إجراء معالجة الحدث Click للزر الذي أنشأناه سابقا وقم بتعديله ليصبح كما في الكود التالي الذي يقوم بتعريف المتغير rd ليكون من النوع ResourceDictionary ثم يستخدم FileStream لفتح الملف CrazyStyle.xaml للقراءة ثم نستخدم XamlReader.Load لقراءة محتويات الملف مستخدمين الدالة CType لتحويل نوع الناتج المقروء إلى ResourceDictionary ثم نقوم بوضع تلك القيمة في المتغير Rd الذي سنضبطه ليكون المصدر الحالي Current Resource للتطبيق وذلك بإسناد قيمته إلى Application.Current.Resources

```
Sub Button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim rd As ResourceDictionary = Nothing
    Using fs As New FileStream("CrazyStyle.xaml", FileMode.Open,
FileAccess.Read)
        rd = CType(XamlReader.Load(fs), ResourceDictionary)
    End Using
    Application.Current.Resources = rd
End Sub
```

شغل التطبيق واختبره

استخدام تحكمات Windows Forms من داخل تطبيق WPF

يمكن استخدام تحكمات Windows Forms من داخل تطبيق WPF من خلال استخدام الفئة WindowsFormsHost Class والتي يكون تعريفها ضمن Visual Basic كالتالي

```
<ContentPropertyAttribute("Child")> _
Public Class WindowsFormsHost _
    Inherits HwndHost _
    Implements IKeyboardInputSink
```

واستخدامها

```
Dim instance As WindowsFormsHost
```

وهي موجودة ضمن مجال الأسماء System.Windows.Forms.Integration وفي المجمع WindowsFormsIntegration وفي المكتبة WindowsFormsIntegration.dll حيث يبين لنا المثال التالي كيفية استخدام العنصر WindowsFormsHost لاستضافة تحكمات Windows Forms

---

```
<Window x:Class="HostingWfInWpf.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
    Title="HostingWfInWpf"
    >
    <Grid>
        <WindowsFormsHost>
            <wf:MaskedTextBox x:Name="mtbDate" Mask="00/00/0000"/>
        </WindowsFormsHost>
    </Grid>
</Window>
```

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
    Title="HostingWfInWpf">
  <Grid>
    <WindowsFormsHost>
      <wf:MaskedTextBox x:Name="mtbDate" Mask="00/00/0000"/>
```

```
    </WindowsFormsHost>

  </Grid>

</Window>
```

ولاستضافة تحكم Windows forms من ضمن تطبيق WPF يجب عليك تحديد تحكم Windows Forms إلى الخاصية Child وذلك باستخدام الخاصية PropertyMap لتحديد الربط بين العنصر WindowsFormsHost و تحكم Windows Forms المستضاف ضمنه

أنشئ تطبيقا من النوع WPF Application و سمه HostingWfInWpf وفي Solution Explorer أضف مرجعا للمجمع WindowsFormsIntegration والمسمى WindowsFormsIntegration.dll ثم قم بفتح Window1.xaml واستبدل كود xaml المولد تلقائيا بالكود التالي

```
<Window x:Class="HostingWfInWpf.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="HostingWfInWpf"
    Loaded="WindowLoaded"
    >
    <Grid Name="grid1">

    </Grid>
</Window>


<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="HostingWfInWpf" Height="300" Width="300"
    Loaded="WindowLoaded"
    >
  <Grid Name="grid1">

    </Grid>
</Window>
```

افتح الملف Window1.xaml.vb في محرر الكود وقم باستبدال جميع الكود الموجود بالكود التالي

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Documents
Imports System.Windows.Media
Imports System.Windows.Media.Imaging
Imports System.Windows.Shapes

Imports System.Windows.Forms

' Interaction logic for Window1.xaml
Partial Public Class Window1
    Inherits Window

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub WindowLoaded(ByVal sender As Object, ByVal e As
RoutedEventArgs)
        ' Create the interop host control.
```

```
        Dim host As New System.Windows.Forms.Integration.WindowsFormsHost()

        ' Create the MaskedTextBox control.
        Dim mtbDate As New MaskedTextBox("00/00/0000")

        ' Assign the MaskedTextBox control as the host control's child.
        host.Child = mtbDate

        ' Add the interop host control to the Grid
        ' control's collection of child controls.
        Me.grid1.Children.Add(host)

    End Sub 'WindowLoaded

End Class
```

اضغط F5 لاختبار المشروع

استخدام عناصر WPF من داخل تطبيق Windows Forms

يمكن استخدام عناصر WPF من داخل تطبيق Windows Forms من خلال استخدام الفئة ElementHost Class والتي يكون تعريفها ضمن Visual Basic كالتالي

```
<ContentPropertyAttribute("Child")> _
Public Class ElementHost
    Inherits Control
```

واستخدامها

```
Dim instance As ElementHost
```

وهي موجودة ضمن مجال الأسماء Namespace المسمى System.Windows.Forms.Integration وفي المجمع WindowsFormsIntegration وفي المكتبة windowsformsintegration.dll ويكون اسم مجال أسماء XML الخاص بها XML Namespace هو http://schemas.microsoft.com/winfx/2006/xaml/presentation ويبين لنا المثال التالي كيفية استخدام التحكم ElementHost لاستضافة عنصر WPF

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' Create the ElementHost control for hosting the
    ' WPF UserControl.
    Dim host As New ElementHost()
    host.Dock = DockStyle.Fill

    ' Create the WPF UserControl.
    Dim uc As New HostingWpfUserControlInWf.UserControl1()

    ' Assign the WPF UserControl to the ElementHost control's
    ' Child property.
    host.Child = uc

    ' Add the ElementHost control to the form's
    ' collection of child controls.
    Me.Controls.Add(host)
End Sub
```

ولاستضافة عنصر WPF ضمن Windows Form يجب عليك تحديد عنصر WPF إلى الخاصية Child وذلك باستخدام الخاصية PropertyMap لتحديد الربط بين التحكم ElementHost وعنصر WPF المستضافة ضمنه كما يمكننا استضافة تحكم Windows Forms ضمن تطبيق WPF باستخدام العنصر WindowsFormsHost

قم بإنشاء مشروع جديد من النوع WPF User Control Library وسمه HostingWpfUserControlInWf ثم قم بفتح الملف UserControl1.xaml في WPF Designer واستبدل كامل الكود الموجود بالكود التالي

```
<UserControl x:Class="HostingWpfUserControlInWf.UserControl1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >

    <Grid>

      <!-- Place a Label control at the top of the view. -->
      <Label
                HorizontalAlignment="Center"
                TextBlock.TextAlignment="Center"
                FontSize="20"
                Foreground="Red"
                Content="Model: Cone"/>

      <!-- Viewport3D is the rendering surface. -->
      <Viewport3D Name="myViewport" >

        <!-- Add a camera. -->
        <Viewport3D.Camera>
          <PerspectiveCamera
                        FarPlaneDistance="20"
                        LookDirection="0,0,1"
                        UpDirection="0,1,0"
                        NearPlaneDistance="1"
                        Position="0,0,-3"
                        FieldOfView="45" />
        </Viewport3D.Camera>

        <!-- Add models. -->
        <Viewport3D.Children>

          <ModelVisual3D>
            <ModelVisual3D.Content>

              <Model3DGroup >
                <Model3DGroup.Children>

                  <!-- Lights, MeshGeometry3D and DiffuseMaterial objects are added to the
                       ModelVisual3D. -->
                  <DirectionalLight Color="#FFFFFFFF" Direction="3,-4,5" />

                  <!-- Define a red cone. -->
                  <GeometryModel3D>

                    <GeometryModel3D.Geometry>
                      <MeshGeometry3D
    Positions="0.293893 -0.5 0.404509  0.475528 -0.5 0.154509  0 0.5 0  0.475528 -0.5
0.154509  0 0.5 0  0 0.5 0  0.475528 -0.5 0.154509  0.475528 -0.5 -0.154509  0 0.5 0
0.475528 -0.5 -0.154509  0 0.5 0  0 0.5 0  0.475528 -0.5 -0.154509  0.293893 -0.5 -0.404509
0 0.5 0  0.293893 -0.5 -0.404509  0 0.5 0  0 0.5 0  0.293893 -0.5 -0.404509  0 -0.5 -0.5  0
0.5 0  0 -0.5 -0.5  0 0.5 0  0 0.5 0  0 -0.5 -0.5  -0.293893 -0.5 -0.404509  0 0.5 0  -
0.293893 -0.5 -0.404509  0 0.5 0  0 0.5 0  -0.293893 -0.5 -0.404509  -0.475528 -0.5 -
0.154509  0 0.5 0  -0.475528 -0.5 -0.154509  0 0.5 0  0 0.5 0  -0.475528 -0.5 -0.154509  -
0.475528 -0.5 0.154509  0 0.5 0  -0.475528 -0.5 0.154509  0 0.5 0  0 0.5 0  -0.475528 -0.5
0.154509  -0.293892 -0.5 0.404509  0 0.5 0  -0.293892 -0.5 0.404509  0 0.5 0  0 0.5 0  -
0.293892 -0.5 0.404509  0 -0.5 0.5  0 0.5 0  0 -0.5 0.5  0 0.5 0  0 0.5 0  0 -0.5 0.5
0.293893 -0.5 0.404509  0 0.5 0  0.293893 -0.5 0.404509  0 0.5 0  0 0.5 0  "
    Normals="0.7236065,0.4472139,0.5257313  0.2763934,0.4472138,0.8506507
0.5308242,0.4294462,0.7306172  0.2763934,0.4472138,0.8506507  0,0.4294458,0.9030925
0.5308242,0.4294462,0.7306172  0.2763934,0.4472138,0.8506507  -0.2763934,0.4472138,0.8506507
0,0.4294458,0.9030925  -0.2763934,0.4472138,0.8506507  -0.5308242,0.4294462,0.7306172
0,0.4294458,0.9030925  -0.2763934,0.4472138,0.8506507  -0.7236065,0.4472139,0.5257313  -
0.5308242,0.4294462,0.7306172  -0.7236065,0.4472139,0.5257313  -0.858892,0.429446,0.279071
-0.5308242,0.4294462,0.7306172  -0.7236065,0.4472139,0.5257313  -0.8944269,0.4472139,0  -
0.858892,0.429446,0.279071  -0.8944269,0.4472139,0  -0.858892,0.429446,-0.279071  -
0.858892,0.429446,0.279071  -0.8944269,0.4472139,0  -0.7236065,0.4472139,-0.5257313  -
0.858892,0.429446,-0.279071  -0.7236065,0.4472139,-0.5257313  -0.5308242,0.4294462,-
0.7306172  -0.858892,0.429446,-0.279071  -0.7236065,0.4472139,-0.5257313  -
0.2763934,0.4472138,-0.8506507  -0.5308242,0.4294462,-0.7306172  -0.2763934,0.4472138,-
0.8506507  0,0.4294458,-0.9030925  -0.5308242,0.4294462,-0.7306172  -0.2763934,0.4472138,-
0.8506507  0.2763934,0.4472138,-0.8506507  0,0.4294458,-0.9030925  0.2763934,0.4472138,-
0.8506507  0.5308249,0.4294459,-0.7306169  0,0.4294458,-0.9030925  0.2763934,0.4472138,-
0.8506507  0.7236068,0.4472141,-0.5257306  0.5308249,0.4294459,-0.7306169
0.7236068,0.4472141,-0.5257306  0.8588922,0.4294461,-0.27907  0.5308249,0.4294459,-0.7306169
0.7236068,0.4472141,-0.5257306  0.8944269,0.4472139,0  0.8588922,0.4294461,-0.27907
0.8944269,0.4472139,0  0.858892,0.429446,0.279071  0.8588922,0.4294461,-0.27907
0.8944269,0.4472139,0  0.7236065,0.4472139,0.5257313  0.858892,0.429446,0.279071
0.7236065,0.4472139,0.5257313  0.5308242,0.4294462,0.7306172  0.858892,0.429446,0.279071  "
```

```
TriangleIndices="0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
59 " />
                    </GeometryModel3D.Geometry>

                    <GeometryModel3D.Material>
                      <DiffuseMaterial>
                        <DiffuseMaterial.Brush>
                          <SolidColorBrush
                            Color="Red"
                            Opacity="1.0"/>
                        </DiffuseMaterial.Brush>
                      </DiffuseMaterial>
                    </GeometryModel3D.Material>

                  </GeometryModel3D>

                </Model3DGroup.Children>
              </Model3DGroup>

            </ModelVisual3D.Content>

          </ModelVisual3D>

        </Viewport3D.Children>

      </Viewport3D>
    </Grid>

</UserControl>
```

- أضف مشروع Windows Forms للـ Solution الحالي وسمه WpfUserControlHost وفي Solution Explorer أضف مرجعا للمجمع WindowsFormsIntegration والمسمى WindowsFormsIntegration.dll ثم قم بإضافة مرجعا Reference لمجمعات WPF التالية PresentationCore و PresentationFramework و WindowsBase ثم أضف مرجعا لـ HostingWpfUserControlInWf ضمن المشروع ثم قم بضبط المشروع WpfUserControlHost ليكون Startup Project
- افتح Form1 في Windows Forms Designer ثم انقر نقرا مزدوجا على form1 لفتح محرر الكود ثم قم باستبدال الكود الموجود بالكود التالي

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Imports System.Windows.Forms.Integration

Public Class Form1
    Inherits Form

    Private Sub Form1_Load(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles MyBase.Load

        ' Create the ElementHost control for hosting the
        ' WPF UserControl.
        Dim host As New ElementHost()
        host.Dock = DockStyle.Fill

        ' Create the WPF UserControl.
        Dim uc As New HostingWpfUserControlInWf.UserControl1()

        ' Assign the WPF UserControl to the ElementHost control's
        ' Child property.
```

```
        host.Child = uc

        ' Add the ElementHost control to the form's
        ' collection of child controls.
        Me.Controls.Add(host)
    End Sub

End Class
```

اضغط F5 لاختبار المشروع

دعنا نقوم بعمل مثال معا خطوة خطوة

أنشئ مشروعا جديدا من النوع Windows Forms Application وسمه TestHostElement

انقر بزر الفأرة اليميني على الـ ToolBox واختر Choose Items من القائمة ومن صفحة net Framework Components. اختر Element Host ليتم إضافتها لـ ToolBox لديك ثم اسحب التحكم ElementHost من الـ ToolBox وضعه على النافذة لديك ثم حدد القيمة Fill للخاصية Dock

في الـ Solution Explorer انقر بزر الفأرة اليميني على المشروع واختر Add New Item ومن صفحة WPF اختر User Control WPF ثم في محرر Xaml الخاص بالـ User Control أضف الكود التالي حتى نقوم بإضافة تحكم Border للتحكم وذلك بعد السطر <Grid> مباشرة حيث تحدد الخاصية Margin الهوامش الخاصة بالتحكم و BorderThikness سماكة الحدود

```
<Border Margin="1,0,0,78" Name="Border1" BorderThickness="0">

</Border>
```

سنقوم بتخصيص شكل الخلفية للتحكم Border وذلك برسم الخلفية الخاصة به باستخدام LinearGradientBrush كما في الكود التالي الذي سنضيفه داخل الكود السابق

```
<Border.Background>
    <LinearGradientBrush StartPoint="0,0.1" EndPoint="0.3,1">
        <GradientStop Offset="0.1" Color="Aquamarine"></GradientStop>
        <GradientStop Offset="0.3" Color="Plum"></GradientStop>
        <GradientStop Offset="0.6" Color="Gold"></GradientStop>
        <GradientStop Offset="0.9" Color="Chocolate"></GradientStop>
        <GradientStop Offset="1" Color="GreenYellow"></GradientStop>
    </LinearGradientBrush>
</Border.Background>
```

أدخل الكود التالي بعد الكود السابق وقبل <Border/> مباشرة وذلك لإضافة تحكم InkCanvas

```
<InkCanvas Margin="5,5,5,5" Name="InkCanvas1" Height="Auto" Width="Auto"
           Background="AliceBlue"/>
```

كي نضيف زر لمسح محتويات InkCanvas أضف الكود التالي بعد <Border/> حيث استخدمنا LinearGradientBrush لرسم الـ Foreground و الـ Background الخاصين بالزر

```
<Button Height="25" Margin="100,0,100,25" Name="btnClear"
        VerticalAlignment="Bottom" Content="Clear InkCanvas">
```

```
    <Button.Foreground>
        <LinearGradientBrush StartPoint="0.3,0.1" EndPoint="0.7,1">
            <GradientStop Offset="0.1" Color="Red"></GradientStop>
            <GradientStop Offset="0.9" Color="Blue" ></GradientStop>
        </LinearGradientBrush>
    </Button.Foreground>
    <Button.Background>
        <LinearGradientBrush StartPoint="0.3,0.1" EndPoint="0.7,1">
            <GradientStop Offset="0.1" Color="#99FFFF"></GradientStop>
            <GradientStop Offset="0.4" Color="#FF99FF"></GradientStop>
            <GradientStop Offset="0.9" Color="#FFFF99" ></GradientStop>
        </LinearGradientBrush>
     </Button.Background>
</Button>
```

من نافذة محرر الـ UserControl انقر نقرا مزدوجا على الزر Clear InkCanvas لإنشاء إجراء معالجة لحدث النقر على ذلك الزر حيث سننتقل هنا مباشرة إلى محرر الكود – أدخل سطر الكود التالي الذي سيقوم بمسح محتويات الـ InkCanvas في الإجراء الفارغ

```
Me.InkCanvas1.Strokes.Clear
```

وبهذا نكون قد انتهينا من تصميم الـ UserControl قم بعمل Build للمشروع

– انتقل الآن إلى محرر تصميم النافذة Form1 واختر التحكم ElementHost وانقر على السهم الصغير الذي يظهر أعلى يمين التحكم لفتح نافذة الخصائص السريعة وافتح القائمة المنسدلة بجانب Select Hosted Content واختر تحكم WPF الذي أنشأناه للتو UserControl1

شغل البرنامج واختبره

كتابة تطبيقنا الأول بتقنية Silverlight

أنشئ تطبيقا جديدا من نوع Silverlight Application وسمه HelloSilverlight ثم اختر Dynamically Generate an HTML test page to host silverlight within this project

تصميم الواجهة

افتح Solution Explorer ولاحظ الملفات app.xaml الذي يحتوي على المصادر والكود الخاص بالتطبيق بأكمله و page.xaml يمثل صفحة شبيهة لتلك الموجودة في موقع ويب وإذا وسعت العقدة بجانبها ستجد ملف page.xaml.vb أو page.xaml.cs بحسب لغة البرمجة التي تستخدمها وهي تحتوي على الكود المدار الخاص بك وهي تشابه للتلك الـ Model الخاصة بـ ASP .net

لتحديد مظهر الشبكة – إن لم تكن الصفحة مفتوحة انقر نقرا مزدوجا على page.xaml لفتحها ثم انتقل لمحرر xaml وحدد عنصر Grid وفي قسم البداية لتعريف الـ Grid غير Background إلى أي لون ترغبه وكذلك حدد الخاصية ShowGridLines إلى true التي تتسبب بإظهار خطوط منقطة توضح أقسام الشبكة الأمر الذي يفيدك عندما تقوم بتصميم الشكل ليظهر القسم كما يلي

```
<Grid x:Name="LayoutRoot" Background="LightGreen"  ShowGridLines="True" >
```

سنستخدم الآن الخاصية RowDefinition للشبكة لتحديد الأسطر والأعمدة

داخل grid و بين قسمي البداية والنهاية لـ Grid أدخل الكود التالي بحيث يبدو تعريف Grid كما يلي

```
<Grid x:Name="LayoutRoot" Background="LightGreen"  ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="220"/>
        <RowDefinition Height="40"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="325"/>
    </Grid.ColumnDefinitions>

</Grid>
```

دعنا نقوم الآن بإضافة بعض التحكمات للتطبيق

قم بسحب كتلة نصوص TextBlock من صندوق الأدوات في بيئة التطوير إلى داخل محرر كود xaml وذلك بعد <Grid.ColumnDefinitions/> مباشرة وأضف له الخاصية Text بالقيمة :Name كما في الكود

```
<TextBlock Text="Name:"></TextBlock>
```

ثم أنشئ كتلتي نصوص أخريين بنفس الطريقة كما في الكود

```
<TextBlock Text="Date:"></TextBlock>
<TextBlock Text="Message"></TextBlock>
```

سترى في نافذة العرض أن صناديق النصوص الثلاثة أصبحت متراكبة في الخلية الأولى في الشبكة ولضبط مكان ظهورها علينا بضبط الخاصيتين Grid.Column ز Grid.row لكل منه كما في الكود حيث ستلاحظ بعد هذه التعديلات ظهور كتل النصوص في أماكنها الصحيحة ضمن الشبكة

```
<TextBlock Text="Name:" Grid.Row="0" Grid.Column="0"> </TextBlock>
<TextBlock Text="Date:" Grid.Row="1" Grid.Column="0"></TextBlock>
<TextBlock Text="Message" Grid.Row="2" Grid.Column="0"
           Grid.ColumnSpan="2" ></TextBlock>
```

اسحب صندوق نصوص textbox من صندوق الأدوات وأفلته مباشرة بعد كتلة النصوص Message واضبط خصائصه كما في الكود

```
<TextBox Text="Your Name" Grid.Row="0" Grid.Column="1"
         Width="150" HorizontalAlignment="Left" ></TextBox>
```

ثم قم بسحب StackPanel إلى محرر xaml أسفل صندوق النصوص والذي يستخدم لصف التحكمات أفقيا أو عموديا داخل خلايا الشبكة واضبط خصائصه كما في الكود

```
<StackPanel Grid.Column="1" Grid.Row="1" Orientation="Vertical" >


</StackPanel>
```

داخل StackPanel أدخل تحكم Calendar وعندما نضيف تحكم التقويم سنلاحظ أن قسم البداية الخاص به مختلف نوعا ما عن بقية التحكمات حيث يبدأ بـ basics أو بادئة أخرى لأنه ليس قسما من تحكمات Silverlight الأساسية وهو معرف ضمن مجمع آخر لذا يجب عليك إضافة مجال أسماء xml ومرجع للمجمع فعندما نقوم بسحب التحكم إلى داخل كود xaml تقوم بيئة التطوير بإضافة المراجع المناسبة تلقائيا من اجلنا الآن قم بضبط خصائص التقويم كما في الكود

```
<basics:Calendar SelectionMode="SingleDate"
                 HorizontalAlignment="Left"></basics:Calendar>
```

اسحب تحكم زر button أسفل تحكم التقويم واضبط خصائصه كما في الكود

```
<Button Width="75" Height="25" HorizontalAlignment="Left"
        Content="OK"></Button>
```

وبهذا نكون قد انتهينا من تصميم الواجهة ويجب أن يبدو كود xaml لديك كما في الكود لا تنس أن تقوم بحفظ المشروع عند هذه النقطة

```
<UserControl xmlns:basics="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls"
x:Class="HelloSilverlight.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="LightGreen"  ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="40"/>
            <RowDefinition Height="220"/>
            <RowDefinition Height="40"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="75" />
            <ColumnDefinition Width="325"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="Name:" Grid.Row="0" Grid.Column="0"> </TextBlock>
        <TextBlock Text="Date:" Grid.Row="1" Grid.Column="0"></TextBlock>
        <TextBlock Text="Message" Grid.Row="2" Grid.Column="0"
                   Grid.ColumnSpan="2" ></TextBlock>
        <TextBox Text="Your Name" Grid.Row="0" Grid.Column="1"
                 Width="150" HorizontalAlignment="Left" ></TextBox>
        <StackPanel Grid.Column="1" Grid.Row="1" Orientation="Vertical" >
            <basics:Calendar SelectionMode="SingleDate"
                             HorizontalAlignment="Left"></basics:Calendar>
            <Button Width="75" Height="25" HorizontalAlignment="Left"
                Content="OK"></Button>
```

```
        </StackPanel>
    </Grid>

</UserControl>
```

إضافة الكود

في محرر xaml انتقل إلى كتلة النصوص Message وفي قسم البداية له أضف الخاصية x:name بالقيمة Message1 وهذه الخاصية تقوم بتعريف العنصر بشكل فريد فيصبح التعريف كما يلي

```
<TextBlock Text="Message" Grid.Row="2" Grid.Column="0"
           x:Name="Message1"></TextBlock>
```

كرر العملية بالنسبة لـ TextBox و Calendare و Button بإضافة الخاصية x:name لها كما يلي

x:Name="name1"

x:Name="cal1"

x:Name="okButton"

في قسم البداية للزر okButton اكتب click ثم اضغط tab فيظهر لك صندوق خاصية Intellisense انقر نقرا مزدوجا على New Event handler فيتم إنشاء إجراء معالجة حدث بالاسم الافتراضي ويوضع في ملف الكود الآن انقر بالماوس اليميني على الحدث click ثم اختر navigate to Event Handler حيث يتم نقلك لمحرر الكود

اجعل إجراء معالجة الحدث يبدو كما في الكود

```
Private Sub okButton_Click(ByVal sender As System.Object, _
                               ByVal e As System.Windows.RoutedEventArgs)

    Dim dateString As String
    If cal1.SelectedDate Is Nothing Then
        dateString = "<date not selected>"
    Else
        dateString = cal1.SelectedDate.ToString()
    End If
    message1.Text = "Hi " & name1.Text & vbCrLf & _
        "Selected Date: " & dateString
End Sub
```

شغل المشروع واختبره

جعل الواجهة ديناميكية

في محرر xaml وضمن قسم البداية لتعريف UserControl احذف الخاصيتين Width و height وفي تعريف Grid.RowDefinitions غير قيم Height كما يلي

<RowDefinition Height="Auto"/>

<RowDefinition Height="*" MinHeight="240"/>

<RowDefinition Height="Auto"/>

وغير خاصية Width لـ Grid.ColumnDefinitions كما يلي

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="*"/>

أضف الخاصية Margin لكل من تحكمات كتل النصوص Name و Date و Message كما يلي

Margin="10,5,10,5"

وللتحكمات TextBox و Calendar و Button كما يلي

Margin="0,5,0,5"

وأضف الخاصية FontSize لكتلة النصوص Message كما يلي

FontSize="20"

احفظ المشروع واختبره ويجب أن يكون لديك كود xaml الكامل كما يلي

```
<UserControl xmlns:basics="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls"
x:Class="HelloSilverlight.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <Grid x:Name="LayoutRoot" Background="LightGreen"  ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="*" MinHeight="240"/>
            <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <TextBlock Text="Name:" Grid.Row="0" Grid.Column="0"
                   Margin="10,5,10,5"> </TextBlock>
        <TextBlock Text="Date:" Grid.Row="1" Grid.Column="0"
                   Margin="10,5,10,5"></TextBlock>
        <TextBlock Text="Message" Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2"
                   x:Name="Message1" Margin="10,5,10,5" FontSize="20"></TextBlock>
        <TextBox Text="Your Name" Grid.Row="0" Grid.Column="1" Margin="0,5,0,5"
                 Width="150" HorizontalAlignment="Left" x:Name="name1"></TextBox>
        <StackPanel Grid.Column="1" Grid.Row="1" Orientation="Vertical" >
            <basics:Calendar SelectionMode="SingleDate" Margin="0,5,0,5"
                             HorizontalAlignment="Left" x:Name="Cal1"></basics:Calendar>
            <Button Width="75" Height="25" HorizontalAlignment="Left"
                Content="OK" x:Name="okButton" Click="okButton_Click"
                    Margin="0,5,0,5"></Button>
        </StackPanel>
    </Grid>

</UserControl>
```

أدوات التحكم بترتيب العناصر Silverlight and WPF

توفر لنا Silverlight و wpf نظام مرن لترتيب العناصر يمكن المطورون والمصممون من تحديد موقع العناصر بسهولة على واجهة المستخدم بحيث يدعم التموضع الثابت والتموضع بواسطة المحددات كما يدعم التموضع الديناميكي الذي يحدد حجم وموضع التحكمات عندما يتغير حجم المستعرض وهنا لدينا ثلاث أدوات تفيدنا في هذا الخصوص هي Canvas و StackPanel و Grid

Canvas Panel

يعتبر الـ Canvas Panel الـ Layout Panel الأساسي الذي يدعم تموضع التحكمات المحتواة فيه باستخدام محددات صريحة ونحن نقوم بتحديد موضع التحكمات فيه باستخدام الخصائص المرتبطة التي تمكنك من تحديد موقعها بالنسبة للتحكم Canvas الأب المباشر لها وهذه الخصائص المرتبطة مفيدة لأنها تمكن اللوح الأب من توسيع الخصائص المضبطة للتحكم الذي بداخله فبتحديد الخصائص المرتبطة مثلا Top و Left الأساسيتين يضيف إمكانية لتحديد ارتباط Top و Left للزر أو أي عنصر آخر ضمن الواجهة بدون الحاجة لإضافة هذه الخاصية لفئة الزر أو تعديل فئة الزر بأي شكل فيمكننا إضافة زرين لـ Canvas وضبط موضعهما بمسافة 50 بكسل من اليسار و 50 و 150 بكسل من الأعلى باستخدام xaml كما في الكود

```
<Canvas Background="Aquamarine" >
    <Button Content="Button 1" Width="100" Height="50"
            Canvas.Left="50" Canvas.Top="50"></Button>

    <Button Content="Button 2" Width="100" Height="50"
            Canvas.Left="50" Canvas.Top="150"></Button>

</Canvas>
```

وبينما يكون الـ Canvas مفيدا في الحالات التي لن يتحرك فيها عناصر الواجهة ولكنك لن تجده بتلك المرونة عندما نضيف تحكمات للواجهة وتعالج أوضاع تحتاج فيها إلى تغيير حجم أو مكان بعض تحكمات الواجهة ففي هذه الحالات سيتوجب عليك كتابة كود النقل أو إعادة التحجيم بنفسك لتحريك الأشياء بداخل الـ Canvas ويكمن الحل الأفضل لهذه المشكلة هو باستخدام عنصر ترتيب مختلف للواجهة مثل StackPanel أو Grid

StackPanel

هو عبارة عن تحكم بسيط لترتيب الواجهة يدعم تموضع التحكمات التي بداخله إما بشكل أفقي أو رأسي وهو يستخدم عادة لترتيب جزء صغير من واجهتك ففي المثال التالي نستخدمه لترتيب ثلاثة أزرار بشكل شاقولي كما في الكود

```
<StackPanel Background="Bisque" >
    <Button Content="Button 1" Width="100" Height="50" Margin="10"/>
    <Button Content="Button 2" Width="100" Height="50" Margin="10"/>
    <Button Content="Button 3" Width="100" Height="50" Margin="10"/>
</StackPanel>
```

أو يمكننا ضبط الخاصية Orientation له إلى Horizontal لضبط الترتيب بشكل أفقي كما في الكود

```
    <StackPanel Background="Bisque" Orientation="Horizontal" >
        <Button Content="Button 1" Width="100" Height="50" Margin="10"/>
        <Button Content="Button 2" Width="100" Height="50" Margin="10"/>
        <Button Content="Button 3" Width="100" Height="50" Margin="10"/>
    </StackPanel>
```

Grid Panel

يعتبر هذا التحكم هو الأكثر مرونة بين الثلاث تحكمات الخاصة بترتيب الواجهة بما انه يدعم ترتيب التحكمات بعدة أسطر وعدة أعمدة وهنا عليك فقط تحديد الأسطر والأعمدة باستخدام الخاصيتين Grid.RowDefinitions و Grid.ColumnDefinitions اللتان يتم تعريفهما

مباشرة تحت التحكم <Grid> ويمكنك استخدام الخصائص المرتبطة Attached Property للتحكمات المحتواة ضمن الـ Grid لتحديد أين يقع التحكم في أي سطر أو عمود فمثلا يمكننا تعريف Grid بثلاثة اسطر و ثلاثة أعمدة ووضع أربع أزرار بداخلها كما في الكود

```
<Grid Background="BlueViolet"
    <Grid.RowDefinitions>
        <RowDefinition Height="60"/>
        <RowDefinition Height="60"/>
        <RowDefinition Height="60"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="110"/>
        <ColumnDefinition Width="110"/>
        <ColumnDefinition Width="110"/>
    </Grid.ColumnDefinitions>
    <Button Content="Button 1" Width="100" Height="50"
            Grid.Column="1" Grid.Row="0">
    </Button>
    <Button Content="Button 2" Width="100" Height="50"
            Grid.Column="0" Grid.Row="1">
    </Button>
    <Button Content="Button 3" Width="100" Height="50"
            Grid.Column="3" Grid.Row="1">
    </Button>
    <Button Content="Button 4" Width="100" Height="50"
            Grid.Column="1" Grid.Row="2">
    </Button>
</Grid>
```

إضافة إلى دعم الـ Grid تحديد الحجم بشكل قيمة ثابتة كما في مثالنا السابق فهي تدعم أيضا التحكم الآلي بالحجم "Height="Auto كما تدعم تحديد حجم أدنى وحجم أقصى بحيث يحدد حجم الأسطر والأعمدة بناء على محتوياتها وحجم الصفحة المتواجدة فيها كما تدعم أن يتم وضع قياس الأسطر والأعمدة نسبة لبعضها ومع استخدامك المتكرر لهذا التحكم ستجد الكثير من المرونة فيه كما تستخدم * لجعل التقسيم بنسب معينة مثل *2 أو *3 كما في المثال

```
<Grid Background="BlueViolet">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="3*"/>
    </Grid.ColumnDefinitions>
    <Button Content="Button 1" Width="100" Height="50"
            Grid.Column="1" Grid.Row="0">
    </Button>
    <Button Content="Button 2" Width="100" Height="50"
            Grid.Column="0" Grid.Row="1">
    </Button>
    <Button Content="Button 3" Width="100" Height="50"
            Grid.Column="3" Grid.Row="1">
    </Button>
    <Button Content="Button 4" Width="100" Height="50"
            Grid.Column="1" Grid.Row="2">
    </Button>
</Grid>
```

إنشاء ساعة تماثلية باستخدام تقنية SilverLight باستخدام الكود فقط

لكي نستطيع العمل سنحتاج إلى Mic rosoft Silver Light Tools For Visual Studio 2008 والذي يمكن تحميله من الرابط
http://go.microsoft.com/fwlink/?LinkId=94863

افتح فيجول ستوديو ومن قائمة File اختر New Project ثم اختر لغة البرمجة Visual Basic ثم اختر Silverlight ثم أنشئ SilverLight Application جديد وقم بتسميته SilverLightClock ومن صندوق الحوار الذي يظهر لنا اختر الآن Automatically Generate a test page to host silverlight at build time ثم اضغط ok

اجعل الكود في قسم xaml مماثلا للكود التالي وذلك لنقوم برسم دائرة رمادية تشكل ظل الساعة حيث استخدمنا Ellips لرسم دائرة وملئها بلون معين

```
<UserControl x:Class="SilverlightClock.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="640" Height="480">

    <Grid x:Name="LayoutRoot">

        <!-- Shadow Ellipse -->
        <Ellipse Margin="165,67,145,83" Fill="#FF000000" Width="330"
         Height="330" Opacity="0.3"/>
    </Grid>
</UserControl>
```

أضف الكود التالي في قسم xaml وقبل </Grid> مباشرة وذلك من أجل رسم الإطار الخارجي للساعة حيث استخدمنا LinearGradientBrush من أجل رسم التدرج اللوني للإطار

```
<!-- Outer Rim -->
<Ellipse Height="330" Margin="156,58,154,92" Width="330" Stroke="#FF000000">
    <Ellipse.Fill>
        <LinearGradientBrush EndPoint="0.84,0.87" StartPoint="0.164,0.129">
            <GradientStop Color="#FFE4E5F4"/>
            <GradientStop Color="#FFC0C0C0" Offset="0.254"/>
        </LinearGradientBrush>
    </Ellipse.Fill>
</Ellipse>
```

أضف الكود التالي بعد الكود السابق في قسم xaml وذلك من أجل رسم حواف إطار الساعة حيث تمت عملية الرسم بنفس الطريقة السابقة

```
<!-- Bevel -->
<Ellipse Height="290" Margin="156,58,154,92" Width="290" Stroke="#FF000000">
    <Ellipse.Fill>
        <LinearGradientBrush EndPoint="0.84,0.87" StartPoint="0.164,0.129">
            <GradientStop Color="#FF2F2F32"/>
            <GradientStop Color="#FFE4E5F4" Offset="0.987"/>
        </LinearGradientBrush>
   </Ellipse.Fill>
</Ellipse>
```

سنستخدم Ellipse مرة أخرى من أجل رسم وجه الساعة – أدخل الكود التالي بعد الكود السابق في قسم xaml

```
<!-- Clock Face -->
<Ellipse Height="270" Margin="176,78,174,112" Width="270"
         Stroke="#FF000000" Fill="#FF000000"/>
```

أدخل الكود التالي في قسم xaml بعد الكود السابق حيث سنستخدم بعضا من أدوات الرسم من أجل عملية رسم العقارب ودائرة المنتصف الخاصة بالساعة

```
<!-- Central Clock Circle -->
<Ellipse Margin="306,208,304,0" VerticalAlignment="Top" Fill="#FF000000"
         Stroke="#FF008000" StrokeThickness="8" Height="30"/>

<!-- Second Hand -->
<Rectangle Height="80" Margin="318.25,117.75,316.75,0" VerticalAlignment="Top"
           Fill="#FFFF0000" Stroke="#FF000000" Width="5"
           RenderTransformOrigin="0.5,1.312" >
    <Rectangle.RenderTransform>
        <RotateTransform x:Name="secondHandTransform"/>
    </Rectangle.RenderTransform>
</Rectangle>

 <!-- Minute Hand -->
<Rectangle x:Name="minuteHand" Height="80" Margin="316.75,117.75,315.25,0"
           VerticalAlignment="Top" Fill="#FF008000" Stroke="#FF008000" Width="8"
           RenderTransformOrigin="0.5,1.312" >
    <Rectangle.RenderTransform>
        <RotateTransform x:Name="minuteHandTransform"/>
    </Rectangle.RenderTransform>
</Rectangle>

<!-- Hour Hand -->
<Rectangle x:Name="hourHand" Height="59" Margin="315.75,138.75,314.25,0"
           VerticalAlignment="Top" Fill="#FF008000" Stroke="#FF008000"
Width="10"
           RenderTransformOrigin="0.525,1.428">
    <Rectangle.RenderTransform>
        <RotateTransform x:Name="hourHandTransform"/>
    </Rectangle.RenderTransform>
</Rectangle>
```

سنضيف الآن حركة الساعة

في قسم xaml بعد تعريف UserControl وقبل <Grid x:Name="LayoutRoot"> أدخل الكود التالي الذي سيقوم بتكوين الحركة الظاهرة للساعة

```
<UserControl.Resources>
    <Storyboard x:Name="clockStoryboard">

        <!-- This animation targets the hour hand transform -->
        <DoubleAnimation x:Name="hourAnimation"
     Storyboard.TargetName="hourHandTransform"
     Storyboard.TargetProperty="Angle"
     Duration="12:0:0" RepeatBehavior="Forever" To="360" />

        <!-- This animation targets the minute hand transform -->
        <DoubleAnimation x:Name="minuteAnimation"
     Storyboard.TargetName="minuteHandTransform"
     Storyboard.TargetProperty="Angle"
     Duration="1:0:0" RepeatBehavior="Forever" To="360" />

        <!-- This animation targets the second hand transform  -->
        <DoubleAnimation x:Name="secondAnimation"
     Storyboard.TargetName="secondHandTransform"
     Storyboard.TargetProperty="Angle"
     Duration="0:1:0" RepeatBehavior="Forever" To="360" />
    </Storyboard>
</UserControl.Resources>
```

فالكود يحدد StoryBoard يحتوي على الرسوم المتحركة التي ستظهر حركة الساعة ولبدء تنفيذه سنستخدم الحدث Loaded لاستدعاء الدالة Begin لـ StoryBoard وسنضيف هذا الحدث لتعريف grid بحيث يصبح كما يلي

```
<Grid x:Name="LayoutRoot" Loaded="SetAndStartClock">
```

وبذلك يكون كود xaml الكامل

```
<UserControl x:Class="SilverLightClock.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="640" Height="480">

    <UserControl.Resources>
        <Storyboard x:Name="clockStoryboard">

            <!-- This animation targets the hour hand transform -->
            <DoubleAnimation x:Name="hourAnimation"
         Storyboard.TargetName="hourHandTransform"
         Storyboard.TargetProperty="Angle"
         Duration="12:0:0" RepeatBehavior="Forever" To="360" />

            <!-- This animation targets the minute hand transform -->
            <DoubleAnimation x:Name="minuteAnimation"
         Storyboard.TargetName="minuteHandTransform"
         Storyboard.TargetProperty="Angle"
         Duration="1:0:0" RepeatBehavior="Forever" To="360" />

            <!-- This animation targets the second hand transform  -->
            <DoubleAnimation x:Name="secondAnimation"
         Storyboard.TargetName="secondHandTransform"
         Storyboard.TargetProperty="Angle"
         Duration="0:1:0" RepeatBehavior="Forever" To="360" />
        </Storyboard>
    </UserControl.Resources>

    <Grid x:Name="LayoutRoot" Loaded="SetAndStartClock">

        <!-- Shadow Ellipse -->
        <Ellipse Margin="165,67,145,83" Fill="#FF000000" Width="330"
         Height="330" Opacity="0.3"/>

        <!-- Outer Rim -->
        <Ellipse Height="330" Margin="156,58,154,92" Width="330" Stroke="#FF000000">
            <Ellipse.Fill>
                <LinearGradientBrush EndPoint="0.84,0.87" StartPoint="0.164,0.129">
                    <GradientStop Color="#FFE4E5F4"/>
                    <GradientStop Color="#FFC0C0C0" Offset="0.254"/>
                </LinearGradientBrush>
            </Ellipse.Fill>
        </Ellipse>

        <!-- Bevel -->
        <Ellipse Height="290" Margin="156,58,154,92" Width="290" Stroke="#FF000000">
            <Ellipse.Fill>
                <LinearGradientBrush EndPoint="0.84,0.87" StartPoint="0.164,0.129">
                    <GradientStop Color="#FF2F2F32"/>
                    <GradientStop Color="#FFE4E5F4" Offset="0.987"/>
                </LinearGradientBrush>
            </Ellipse.Fill>
        </Ellipse>


        <!-- Clock Face -->
        <Ellipse Height="270" Margin="176,78,174,112" Width="270"
                 Stroke="#FF000000" Fill="#FF000000"/>


        <!-- Central Clock Circle -->
        <Ellipse Margin="306,208,304,0" VerticalAlignment="Top" Fill="#FF000000"
                 Stroke="#FF008000" StrokeThickness="8" Height="30"/>
```

```
        <!-- Second Hand -->
        <Rectangle Height="80" Margin="318.25,117.75,316.75,0" VerticalAlignment="Top"
                   Fill="#FFFF0000" Stroke="#FF000000" Width="5"
                   RenderTransformOrigin="0.5,1.312" >
            <Rectangle.RenderTransform>
                <RotateTransform x:Name="secondHandTransform"/>
            </Rectangle.RenderTransform>
        </Rectangle>

        <!-- Minute Hand -->
        <Rectangle x:Name="minuteHand" Height="80" Margin="316.75,117.75,315.25,0"
                   VerticalAlignment="Top" Fill="#FF008000" Stroke="#FF008000" Width="8"
                   RenderTransformOrigin="0.5,1.312" >
            <Rectangle.RenderTransform>
                <RotateTransform x:Name="minuteHandTransform"/>
            </Rectangle.RenderTransform>
        </Rectangle>

        <!-- Hour Hand -->
        <Rectangle x:Name="hourHand" Height="59" Margin="315.75,138.75,314.25,0"
                   VerticalAlignment="Top" Fill="#FF008000" Stroke="#FF008000" Width="10"
                   RenderTransformOrigin="0.525,1.428">
            <Rectangle.RenderTransform>
                <RotateTransform x:Name="hourHandTransform"/>
            </Rectangle.RenderTransform>
        </Rectangle>

    </Grid>
</UserControl>
```

من Solution Exploere وسع عقدة Page.xaml ثم افتح الملف Page.xaml.vb وأدخل فيه الكود التالي من أجل بدء عملية التحريك

```
    Private Sub SetAndStartClock(ByVal sender As Object, ByVal e As EventArgs)

        ' Start the storyboard.
        clockStoryboard.Begin()
    End Sub
```

الآن ومع أن الساعة أصبحت تعمل بالشكل المطلوب إلا أنها لاتظهر الوقت الصحيح الأن استبدل الكود السابق بالكود التالي

```
Private Sub SetAndStartClock(ByVal sender As Object, ByVal e As EventArgs)

    ' The current date and time.
    Dim currentDate As Date = DateTime.Now

    ' Find the appropriate angle (in degrees) for the hour hand
    ' based on the current time.
    Dim hourangle As Double = (((CType(currentDate.Hour, Single) / 12) * 360) +  _
                              (currentDate.Minute / 2))

    ' The same as for the minute angle.
    Dim minangle As Double = ((CType(currentDate.Minute, Single) / 60) * 360)

    ' The same for the second angle.
    Dim secangle As Double = ((CType(currentDate.Second, Single) / 60) * 360)

    ' Set the beginning of the animation (From property) to the angle
    ' corresponging to the current time.
    hourAnimation.From = hourangle

    ' Set the end of the animation (To property)to the angle
    ' corresponding to the current time PLUS 360 degrees. Thus, the
    ' animation will end after the clock hand moves around the clock
    ' once. Note: The RepeatBehavior property of the animation is set
```

```
    ' to "Forever" so the animation will begin again as soon as it completes.
    hourAnimation.To = (hourangle + 360)

    ' Same as with the hour animation.
    minuteAnimation.From = minangle
    minuteAnimation.To = (minangle + 360)

    ' Same as with the hour animation.
    secondAnimation.From = secangle
    secondAnimation.To = (secangle + 360)

    ' Start the storyboard.
    clockStoryboard.Begin()

End Sub
```

احفظ المشروع واعمل Build ثم جربه يجب أن تعمل الساعة الآن بشكل صحيح

الفروقات في معالجة xaml بين Silverlight و WPF

يستخدم Silverlight تعريفا محددا لمعرب xaml وبما أن هذا المعرب هو جزء من مكتبات الزبون العائدة لـ Silverlight لهذا قد يختلف تصرف الإعراب بينها وبين WPF والذي يمتلك أيضا تعريفا محددا وقد تكون الفروقات المذكورة هنا مفيدة عندما تقوم بتهجير كود xaml مكتوب أصلا من أجل WPF إلى Silverlight

الفروقات في معالجة xaml بين WPF و Silverlight

مجالات الأسماء

- يدعم سيلفرلايت http://schemas.microsoft.com/client/2007 إضافة إلى http://schemas.microsoft.com/client/2006/xaml/presentation كمجال الأسماء الزبون
- يضع سيلفرلايت القيود التالية على قيم xaml
  - العنصر الجذري يجب دوما أن يحتوي على تعريف xaml الافتراضي
  - أي تعريف افتراضي يجب أن يكون http://schemas.microsoft.com/client/2007 أو http://schemas.microsoft.com/client/2006/xaml/presentation أو مجال أسماء XPS
  - أي تعريف غير افتراضي لا يستخدم احد المجالين المذكورين سابقا غير مسموح به
- يقدم سيلفرلايت القيود التالية على تنظيم المجمعات ومجالات الأسماء من أجل قيم xaml
  - يجب على المجمع إما أن يكون mscorlib أو أن يكون اسم المجمع في ملف xap فلا يمكن أن يكون المجمع ملف DLL من تنصيب سيلفرلايت الأساسي ولا حتى مكتبة خارجية كلية
  - لايمكن أن يتضمن اسم المكتبة في نهايتها dll.

البانيات

- الباني الوحيد المدعوم من سيلفرلايت هو xml:lang
- البانيات :x المدعومة من قبل سيلفرلايت هي x:Null و x:Name و x:Key و x:Class
- بانيات :sys المدعومة هي sys:Double و sys:String و sys:int32
- باني :mc الوحيد المدعوم هو mc:Ignorable

لغة xaml الجوهرية

- هناك دعم محدود لـ Markup Extensions في سيلفرلايت والـ Markup Extensions المدعومة هي x:Type و x:Null و StaticResource و Binding و TemplateBindin g ولا يوجد Markup Extensions عامة إلا فيما يتعلق بـ Binding مما يعني أنه لا يوجد Markup Extensions مخصصة
- فيما عدا Binding تستخدم الـ Markup Extensions بصيغة الخصائص attributes فلا يوجد صيغة لعنصر الغرض هنا ومحددات البانيات لا يمكن تسميتها فمثلا {StaticResource aKey} صحيحة ولكن {StaticResource ResourceKey=aKey} خاطئة
- لا تدعم سيلفرلايت إضافة extension إلى نهاية اسم Markup Extension كاستخدام بديل فمثلا {x:Null} صحيح بينما {x:NullExtension} خاطئ
- في شروط تصرفات xaml هناك دعم محدود في سيلفرلايت لعناصر الأغراض التي ليست DependencyObject
  - عناصر الأغراض التي ليست DependencyObject لا تدعم x:Name
  - الفئات التالية التي ليست DependencyObject مدعومة كالفئات المعرفة في المجمع والتي تم تحميلها مع التطبيق وفئات :sys هي Color و FontFamily
- الخاصية Name مدعومة من قبل جميع الفئات الفرعية في سيلفرلايت وتعامل مثل x:Name وفي الفئات الفرعية FrameworkElement المعرب سيعدل قيمة الخاصية Name وحتى إن لم يكن لها رابط عام في الكود
- العناصر في ResourceDicitonary في سيلفرلايت ربما يكون لها x:Name بدلا عن أو إضافة إلى x:Key وإن لم تكن x:Key محددة يستخدم x:Name بدلا عنها

- باستثناء TextBlock و Run عناصر الأغراض في سيلفرلايت لايمكن أن تحتوي على عقد نص xml كطريقة لتعريف نص المحتويات للغرض فمثلا <Button>Hello World</Button> غير مسموح به

التصرفات الأخرى

- الـ Framework Template وفئاتها الفرعية تدعم الخاصية content حتى ولوم لم يكن لها الخاصية Content
- UserControl.Content هي خاصية محمية على العكس من الخصائص العامة وكتصرف عام يمكن لمعرب سيلفرلايت أن يضبط هذه المحتويات طالما أن قيمة x:Class تم تحديدها

كيف نستخدم عناصر Style للتحكم بمظهر التطبيق

قمنا في موضوع سابق لي بتغيير شكل الزر بواسطة كود Style في الملف App.xaml وكان الكود هو

```
<Style x:Key="RoundButton" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
              <Grid>
                    <Ellipse Width="{TemplateBinding Width}"
                             Height="{TemplateBinding Height}">
                        <Ellipse.Fill>
                            <RadialGradientBrush GradientOrigin=".2,.2">
                                <GradientStop Offset=".2" Color="White"/>
                                <GradientStop Offset="1" Color="Blue" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <ContentPresenter  Content="{TemplateBinding Content}"
                                       HorizontalAlignment="Center"
                                       VerticalAlignment="Center" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

وقمنا بربطه بالزر حتى أخذ الشكل الجديد بواسطة الكود

```
<Button Width="200" Height="200" Content="Hello VB"
        Style="{StaticResource RoundButton}" FontSize="15">

</Button>
```

حيث تزودنا Silverlight و WPF بآلية دعم للأشكال تمكننا من تغليف خصائص التحكم واستخدامها كمصادر قابلة لإعادة الاستخدام مما يمكننا من تخزين هذه المصادر في ملفات منفصلة عن صفحتنا وإعادة استخدامها عبر العديد من التحكمات والصفحات في التطبيق وإضافة إلى إمكانية تحديد الخصائص الأساسية يمكن لهذه الأشكال Style أن تعيد استخدام قوالب الأشكال الخاصة بالتحكمات مما يمكننا من إعادة تشكيل كامل مظهر هذه التحكمات بالشكل الذي نريده

فإن كان لدينا تحكمان Border و TextBlock تم تعريفهما كما يلي

```
<Border Grid.Column="0" Grid.Row="0" CornerRadius="30" Background="Blue"
        Margin="10" Height="50">
    <TextBlock Text="Test" Foreground="Yellow" Margin="3" FontSize="20"
             HorizontalAlignment="Center" VerticalAlignment="Center" ></TextBlock>
</Border>
```

فيمكننا إنشاء شكل Style ضمن الملف app.xaml ليغلف إعدادات كلا التحكمين كما يلي

```
<Style x:Key="MyBorder" TargetType="Border">
    <Setter Property="CornerRadius" Value="30"/>
    <Setter Property="Background" Value="Blue"/>
    <Setter Property="Margin" Value="10"/>
    <Setter Property="Height" Value="50"/>
</Style>

<Style x:Key="MyTextBlock" TargetType="TextBlock">
    <Setter Property="Foreground" Value="Yellow"/>
    <Setter Property="FontSize" Value="20"/>
    <Setter Property="Margin" Value="3"/>
```

```
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>
```

حيث أصبح الآن بإمكاننا إعادة استخدام هذه الأشكال مع أكثر من تحكم كما يلي

```
<Border Grid.Column="0" Grid.Row="0" Style="{StaticResource MyBorder}">
    <TextBlock Text="Test" Style="{StaticResource MyTextBlock}"></TextBlock>
</Border>
```

كما يمكننا استخدام Style للتحكم بمظهر العديد من التحكمات الأخرى الشبكة مثلا

```
<Style x:Key="TopGrid" TargetType="Grid">
      <Setter Property="Background" Value="#FF5C7590" />
</Style>
```

أو StackPanel

```
<Style x:Key="DiggPanel" TargetType="StackPanel">
    <Setter Property="Margin" Value="10"/>
    <Setter Property="Width" Value="55"/>
    <Setter Property="Height" Value="55"/>
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop Color="#FFFFF098"/>
                <GradientStop Color="#FFFFF9D4" Offset="1"/>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>
```

أو الزر

```
<Style x:Key="CloseButton" TargetType="Button">
    <Setter Property="HorizontalAlignment" Value="Right"/>
    <Setter Property="Width" Value="50"/>
    <Setter Property="Height" Value="25"/>
    <Setter Property="Template">
        <Setter.Value>
                <ControlTemplate>
                    <Border x:Name="brd1" Width="22" Height="22" CornerRadius="15">
                    <TextBlock x:Name="txt1" Foreground="#222" TextAlignment="center"
                Text="r" FontSize="11" VerticalAlignment="center" FontFamily="Webdings"/>
                        <Border.Background>
                            <RadialGradientBrush GradientOrigin=".3, .3">
                                <GradientStop Color="#FFF" Offset=".15"/>
                                <GradientStop Color="#777" Offset="1"/>
                            </RadialGradientBrush>
                        </Border.Background>
                    </Border>
                </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

أو HyperLinkButton

```
<Style x:Key="TitleLink" TargetType="HyperlinkButton">
    <!--<Setter Property="TextWrapping" Value="Wrap"/>-->
    <Setter Property="HorizontalAlignment" Value="Left"/>
    <Setter Property="FontSize" Value="16"/>
    <Setter Property="Foreground" Value="White"/>
```

```
    <Setter Property="Width" Value="500"/>
    <Setter Property="Grid.Row" Value="0"/>
    <Setter Property="Grid.Column" Value="1"/>
    <Setter Property="Grid.ColumnSpan" Value="2"/>
</Style>
```

وبنفس الأسلوب يمكننا تخصيص مظهر أي تحكم نريده

سؤال

أين نضع الاستيل في نفس الملف أم في ملف ResourceDictionary ؟

الجواب

يتم وضع الاستايل في الملف App.xaml وذلك بعد <Application.Resources> و قبل </Application.Resources> مثال

```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="SilverlightApplication5.App"
             >
    <Application.Resources>

        <Style x:Key="MyBorder" TargetType="Border">
            <Setter Property="CornerRadius" Value="30"/>
            <Setter Property="Background" Value="Blue"/>
            <Setter Property="Margin" Value="10"/>
            <Setter Property="Height" Value="50"/>
        </Style>

        <Style x:Key="MyTextBlock" TargetType="TextBlock">
            <Setter Property="Foreground" Value="Yellow"/>
            <Setter Property="FontSize" Value="20"/>
            <Setter Property="Margin" Value="3"/>
            <Setter Property="HorizontalAlignment" Value="Center"/>
            <Setter Property="VerticalAlignment" Value="Center"/>

        </Style>

    </Application.Resources>
</Application>
```

تخصيص مظهر التحكمات Silverlight & WPF

إحدى أكثر الميزات قوة في SLiverlight و WPF هي الإمكانية الكاملة لتخصيص مظهر التحكمات التي نستخدمها مما يمكن المطورون والمصممون لنحت واجهة التحكمات بالشكل الذي يريدونه مما يوفر لهم مرونة كبيرة لتحقيق الواجهة التي يرغبون بها حيث سنقوم باستعراض هذه الامكانية في مقالتنا هذه حول تحكم الزر Button من أجل استعراض بعض ما يمكننا فعله مع أن كثيرا من التحكمات الأخرى يمكن أن يتم تعديل مظهرها بطريقة مشابهة لما سنقوم به هنا

لنفرض أنه لدينا في أحد التطبيقات زر تم تعريفه في قسم XAML كما يلي

```
<Button x:Name="btnTest" Content="Push ME!" Width="100" Height="50"
        Click="btnTest_Click">


</Button>
```

أحد الأشياء التي تفيدنا هو أن الخاصية Contenet لا يجب أن تكون نصية دوما بل يمكنها أن تكون أية سلسلة من الأشكال والتحكمات التي نرغب باستخدامها فقد نرغب بجعلها تتضمن StackPanel يتضمن Image و TextBlock بداخله كما في المثال

```
<Button x:Name="btnTest" Width="200" Height="100"
        Click="btnTest_Click">
    <Button.Content>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <Image Source="Samer.jpg" Height="75"></Image>
            <TextBlock Text="Samer" FontSize="20" VerticalAlignment="Center"
                       Margin="15,10,10,10" ></TextBlock>
        </StackPanel>
    </Button.Content>

</Button>
```

أو يمكننا استخدام تحكمات الأشكال لرسم الأشكال الخاصة بنا مثل تحكم Ellipse مثلا كما في الكود

```
<Button x:Name="btnTest" Width="200" Height="100"
        Click="btnTest_Click">
    <Button.Content>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <Ellipse Margin="10" Width="50" Height="50">
                <Ellipse.Fill>
                    <RadialGradientBrush GradientOrigin="0.2,0.2">
                        <GradientStop Offset="0.2" Color="White" />
                        <GradientStop Offset="1" Color="Blue" />
                    </RadialGradientBrush>
                </Ellipse.Fill>
            </Ellipse>
            <TextBlock Text="Samer" FontSize="20" VerticalAlignment="Center"
                       Margin="15,10,10,10" ></TextBlock>
        </StackPanel>
    </Button.Content>

</Button>
```

كما يمكننا تضمين تحكمات مثل Calendar Controls ضمن الزر حيث يمكن للمستخدم التنقل ضمن تحكم التقويم وتحديد التاريخ الذي يرغب به ثم القيام بضغط الزر الذي يحتويه كما في الكود

```
<Button x:Name="btnTest" Width="400" Height="200"
```

```
        Click="btnTest_Click">
    <Button.Content>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <basics:Calendar></basics:Calendar>
            <TextBlock Text="Samer" FontSize="20" VerticalAlignment="Center"
                       Margin="15,10,10,10" ></TextBlock>
        </StackPanel>
    </Button.Content>

</Button>
```

كما تمكننا الامكانيات البرمجية في Silverlight و wpf من القيام بأشياء أكثر من مجرد تعديل المحتويات الداخلية للتحكم حيث لدينا الإمكانية هنا لاستبدال كامل مظهر التحكم بأي شئ نرغبه بدون أن نؤثر على التصرف الطبيعي للتحكم فعلى سبيل المثال نريد من أزار برنامجنا أن تأخذ شكلا مستديرا عوضا عن الشكل المستطيل التقليدي فهنا يمكننا عمل ذلك بإنشاء Style يدعى RoundButton سنستخدمه لنتجاوز الخاصية Template للأزار وتقديم Template أخرى تستبدل الشكل المستطيل التقليدي بتحكم Ellipse يحتوي بداخله TextBlock

في مشروع Sliverlight لديك افتح الملف App.xaml وأضف الكود التالي الذي ينشئ الشكل الذي نريد استخدامه مع الأزار لدينا وذلك ضمن القسم <Application.Resources> ثم اعمل Build للمشروع

```
<Style x:Key="RoundButton" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                 <Grid>
                    <Ellipse Width="200" Height="200">

                        <Ellipse.Fill>
                            <RadialGradientBrush GradientOrigin=".2,.2">
                                <GradientStop Offset=".2" Color="White"/>
                                <GradientStop Offset="1" Color="Blue" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <TextBlock Text="Push Me!"
                               FontSize="28"
                               HorizontalAlignment="Center"
                               VerticalAlignment="Center" />
                 </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
 </Style>
```

الآن أضف الخاصية Style للزر الذي نرغب بتغيير شكله وذلك بالقيمة {StaticResource RoundButton} كما في المثال

```
<Button Width="200" Height="200" Style="{StaticResource RoundButton}">

</Button>
```

نلاحظ هنا أن النص المعروض في الزر دوما هو Push Me! وللتغلب على هذه المشكلة وجعل كل زر يعرض النص الذي نرغب به نستخدم الخاصية TemplateBinding لربط خصائص التحكم مع الشكل الذي قمنا بإنشائه حيث سنقوم بتعديل الشكل السابق كما يلي

```
<Style x:Key="RoundButton" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
```

```
            <Grid>
                  <Ellipse Width="{TemplateBinding Width}"
                           Height="{TemplateBinding Height}">
                      <Ellipse.Fill>
                          <RadialGradientBrush GradientOrigin=".2,.2">
                              <GradientStop Offset=".2" Color="White"/>
                              <GradientStop Offset="1" Color="Blue" />
                          </RadialGradientBrush>
                      </Ellipse.Fill>
                  </Ellipse>
                  <ContentPresenter  Content="{TemplateBinding Content}"
                                     HorizontalAlignment="Center"
                                     VerticalAlignment="Center" />
              </Grid>
          </ControlTemplate>
      </Setter.Value>
  </Setter>
</Style>
```

اعمل build للمشروع بعد تعديل الشكل ثم قم بضبط الخصائص المرغوبة من تحكم الزر كما في الكود

```
<Button Width="200" Height="200" Content="Hello VB"
        Style="{StaticResource RoundButton}" FontSize="15">

</Button>
```

يمكنك الملاحظة من الكود السابق للشكل أننا قمنا باستبدال TextBlock بـ ContenetPresenter وذلك من أجل أن نتمكن من عرض أي شكل نريده داخل الزر وليس النصوص فقط وإن أردنا المزيد من التحكم بمظهر الزر يمكننا أن نطوره ليعالج حالات الزر المختلفة مثل hover أو focus أو Pushed مما يمكننا من تشكيل الأشكال التي نريدها تماما

القسم الرابع - النظام والملفات

ويضم المواضيع التالية:

- مثال على عملية إنشاء Windows Service إنشاء برنامج تشفير تلقائي للملفات
- كيف يمكننا التأكد من تحرير موارد النظام التي يستخدمها كودنا
- كيفية إضافة بنود إلى قائمة النظام برمجيا
- كيف يمكننا البحث عن ملف بمحتوى معين ضمن شجرة مجلدات
- كيف نقوم بالبحث في سجل النظام
- تعالوا نعمل معا Task Manager بسيط وبسرعة
- تعقب إضافة وإزالة الأقراص المرتبطة عبر منفذ Usb
- تشغيل برنامج من ضمن كود فيجول بايزيك دوت نيت
- الأصوات في VB.net
- إدخال وإخراج الأقراص القابلة للنزع برمجيا
- مراقبة نظام الملفات - التحكم FileSystemWatcher

مثال على عملية إنشاء Windows Service – إنشاء برنامج تشفير تلقائي للملفات

فكرة المشروع – بناء خدمة ويندوز تقوم بتشفير أي ملف تلقائيا عند نسخه إلى مجلد معين وسنستخدم هنا إجرائية تشفير قابلة للعكس بحيث أن عملية إعادة نسخ الملف لذلك المجلد تقوم بفك تشفيره

أنشئ مشروعا جديدا من نوع Windows Service وقم بتسميته AutoFileEncryptor وغير تسمية Service1.vb إلى AutoFileEncryptor.vb ووافق على الرسالة التي تطلب منك إعادة تسمية الفئة ثم افتح محرر التصميم لـ AutoFileEncryptor و اسحب العنصر FileSystemWatcher على سطح النافذة ثم افتح محرر الكود الخاص بـ AutoFileEncryptor

أضف التعريف التالي في القسم العام للفئة AutoFileEncryptor

```
Private WatcPath = "d:\AutoFileEncryptor"
```

عدل إجراء الحدثين OnStart و OnStop ليبدوا كما في الكود التالي وذلك لتعيين الخصائص الابتدائية للعنصر FileSystemWatcher

```
Public Class AutoFileEncryptor
    Private WatcPath = "d:\AutoFileEncryptor"

    Protected Overrides Sub OnStart(ByVal args() As String)
        ' Add code here to start your service. This method should set things
        ' in motion so your service can do its work.
        Me.FileSystemWatcher1.Path = WatcPath
        Me.FileSystemWatcher1.EnableRaisingEvents = True
    End Sub

    Protected Overrides Sub OnStop()
        ' Add code here to perform any tear-down necessary to stop your service.
        Me.FileSystemWatcher1.EnableRaisingEvents = False
    End Sub

End Class
```

لكتابة إجرائية التشفير سنحتاج أولا لإضافة التعريفات العامة التالية في فئتنا

```
    ' This is the binary password.
    Dim pwBytes() As Byte = {123, 234, 12, 9, 78, 89, 212}
    ' This is the extension used for temporary files.
    Dim tempExt As String = ".$$$"
```

وأيضا أضف الاستيراد التالي في بداية الملف

```
Imports System.IO
```

ضمن الفئة اكتب الكود التالي الذي يمثل إجرائية التشفير

```
    ' This is the encryption/decryption routine.
    Private Sub EncryptFile(ByVal Filename As String, ByVal pwBytes() As Byte)
        ' This is the size of each input block.
        ' (Files must be decrypted using the same block size.)
        Const BLOCKSIZE = 8192
        ' Determine the name of the temporary file.
        Dim tempFile As String = Filename & tempExt
        ' Open the source file as a binary input stream.
        Dim inStream As New FileStream(Filename, IO.FileMode.Open)
        ' Open the temporary output file as a binary input stream.
        Dim outStream As New FileStream(tempFile, IO.FileMode.Create)
        ' Determine the number of bytes to read.
        Dim bytesLeft As Long = inStream.Length
        ' Prepare an input buffer.
        Dim buffer(BLOCKSIZE - 1) As Byte
        ' Loop until there are bytes to read.
```

```
        Do While bytesLeft > 0
            ' Read max 8 KB at a time.
            Dim bytesToRead As Long = Math.Min(BLOCKSIZE, bytesLeft)
            ' Read into the input buffer.
            inStream.Read(buffer, 0, bytesToRead)
            ' Encrypt this buffer.
            EncryptArray(buffer, pwBytes)
            ' Output to the temporary file.
            outStream.Write(buffer, 0, bytesToRead)
            ' We have fewer bytes to read now.
            bytesLeft -= bytesToRead
        Loop
        ' Close the two streams.
        inStream.Close()
        outStream.Close()
        ' Delete the source file.
        File.Delete(Filename)
        ' Rename the temporary file as the original file.
        File.Move(tempFile, Filename)
    End Sub

    ' This routine encrypts an array of bytes.
    Sub EncryptArray(ByVal buffer() As Byte, ByVal pwBytes() As Byte)
        ' This index points to the password array.
        Dim i As Integer
        ' The max value for i
        Dim maxval As Integer = pwBytes.Length
        For index As Integer = 0 To buffer.Length - 1
            ' XOR each element with the corresponding element in the password.
            buffer(index) = buffer(index) Xor pwBytes(i)
            ' Ensure that the index is always in the valid range.
            i = (i + 1) Mod maxval
        Next
    End Sub
```

أضف إجراء معالجة للحدث Created الخاص بالعنصر FileSystemWatcher وعدله ليصبح كما في الكود التالي

```
    Private Sub FileSystemWatcher1_Created(ByVal sender As Object, _
                ByVal e As System.IO.FileSystemEventArgs) _
                Handles FileSystemWatcher1.Created

        ' Ignore temporary files created by the encryption process.
        If System.IO.Path.GetExtension(e.FullPath) = tempExt Then Exit Sub
        ' Encrypt the file being created.
        EncryptFile(e.FullPath, pwBytes)
    End Sub
```

أضف الكود التالي للإجراء OnStart كي نتأكد عند بداية الخدمة أن المجلد موجود فعلا

```
If Not Directory.Exists(WatcPath) Then
    Directory.CreateDirectory(WatcPath)
End If
```

فيصبح الإجراء كاملا

```
    Protected Overrides Sub OnStart(ByVal args() As String)
        ' Add code here to start your service. This method should set things
        ' in motion so your service can do its work.
        If Not Directory.Exists(WatcPath) Then
            Directory.CreateDirectory(WatcPath)
        End If
        Me.FileSystemWatcher1.Path = WatcPath
        Me.FileSystemWatcher1.EnableRaisingEvents = True
```

```
End Sub
```

انتقل إلى سطح التصميم لـ AutoFileEncryptor واضبط الخاصية ServiceName لـ AutoFileEncryptor إلى Auto File Encryptor واضغط زر الفأرة اليميني واختر الأمر Add Installer من قائمة السياق ومن خصائص ServiceInstaller1 اضبط كلتا الخاصيتين DisplayName و ServiceName إلى Auto File Encryptor ثم انتقل لخصائص ServiceProcessInstaller1 واضبط الخاصية Account إلى Local Service ثم اختر الأمر Save All من قائمة File

قم بعمل Build للمشروع

أنشئ مجلدا مؤقتا باسم Temp على السواقة D وانسخ الملف التنفيذي للمشروع إليه

ثم أنشئ فيه ملفا نصيا باسم Install.bat بحيث تكون محتوياته

```
INSTALLUTIL AutoFileEncryptor.exe
```

وأيضا ملف نصي آخر باسم Uninstall.bat بحيث تكون محتوياته

```
INSTALLUTIL /U AutoFileEncryptor.exe
```

انتقل إلى مجلد Visual Studio في قائمة ابدأ و شغل الأمر Visual Studio 2008 Command Prompt وتأكد من أنك شغلته باستخدام الخيار Run as Administrator عندما تعمل على ويندوز فيستا ثم من نافذة الكونسول اكتب الأوامر التالية تباعا مع الضغط على Enter بعد كل سطر

```
D:

CD\temp

Install
```

فيتم تنصيب الخدمة في ويندوز

اترك نافذة الكونسول مفتوحة وانتقل الآن إلى بيئة التطوير وانتقل إلى نافذة Server Explorer ووسع الشجرة حتى ترى العقدة Services انقر على العقدة Services بزر الفأرة اليميني و اختار الأمر Launch Services Manager ابحث ضمن القائمة عن Auto File Encryptor ثم قم ببدء تشغيل الخدمة

لتجربة الخدمة انسخ أي ملف نصي إلى المجلد d:\AutoFileEncryptor وحاول قراءة محتويته ستراها بالصيغة المشفرة ولفك تشفير الملف انقله إلى أي مجلد آخر ثم انسخه للمجلد d:\AutoFileEncryptor مرة ثانية ليتم فك تشفيره تلقائيا

لإزالة الخدمة عد إلى برنامج Services Manager وقم بإيقاف الخدمة ثم عد إلى نافذة الكونسول ونفذ الأمر Uninstall

كيف يمكننا التأكد من تحرير موارد النظام التي يستخدمها كودنا

يمكنك أن تستخدم بلوك using من أجل التأكد من أن كودك قد تخلص تماما من مصادر النظام التي يستخدمها وخصوصا عندما تستخدم أوامر تتطلب استخدام قدرا كبيرا من الذاكرة وذلك بغض النظر عن الطريقة التي استخدمتها للخروج من بلوك using حتى لو أطلق الكود استثناء غير معالج . فمثلا كي نتأكد من أننا قمنا بالتخلص من كامل كمية الذاكرة التي يستهلكها الاتصال بقاعدة البيانات يمكننا تضمينه ضمن بلوك using ونكتب الكود الذي يتعامل مع قاعدة البيانات داخل بلوك using كما في المثال

```
Imports System.Data.SqlClient

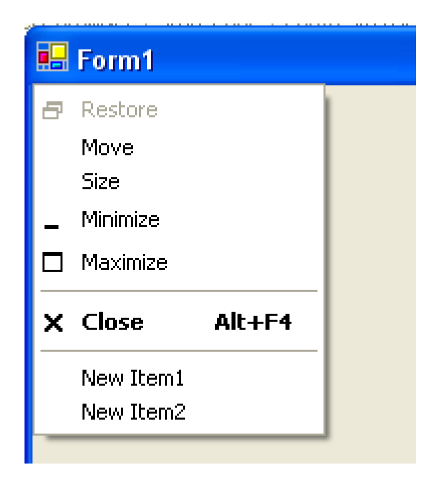
Public Class Form1

    Public Sub AccessSql(ByVal s As String)
        Using sqc As New System.Data.SqlClient.SqlConnection(s)
            MsgBox("Connected with string """ & sqc.ConnectionString & """")
        End Using
    End Sub

End Class
```

مع ملاحظة أن المتغير الذي تم تعريفه عند التصريح عن بلوك using مثل المتغير sqc في مثالنا هنا لا يمكن الوصول إليه من خارج بلوك using لأنه يتم التخلص منه فور الخروج من البلوك كما يمكنك استخدام هذا البلوك من أجل العديد من مصادر النظام مثل فتح منفذ تسلسلي أو الكتابة إلى ملف وبشكل عام أنت تستخدم هذا البلوك عندما تريد أن تتأكد أن المصدر الذي تستخدمه يجب أن يكون متاحا فور الانتهاء منه . مع العلم أن المتغير الذي يعرف باستخدام Using يجب أن يضمن الواجهة IDisposable

كيفية إضافة بنود إلى قائمة النظام برمجيا



أولا – في قسم الإعلانات العامة نضيف التعريفات التالية:

```
Private Declare Function GetSystemMenu Lib "user32" (ByVal hwnd As Integer, _
    ByVal bRevert As Boolean) As Integer

Private Declare Function AppendMenu Lib "user32" Alias "AppendMenuA" _
    (ByVal hMenu As Integer, ByVal wFlags As Integer, ByVal wIDNewItem As Integer _
    , ByVal lpNewItem As String) As Integer

Private Declare Function RemoveMenu Lib "user32" (ByVal hMenu As Long, _
        ByVal nPosition As Long, ByVal wFlags As Long) As Long

Const MF_BYPOSITION = &H400&
Const MF_REMOVE = &H1000&
Const MF_SEPARATOR = &H800&
Const WM_SYSCOMMAND As Integer = &H112
```

ثانيا – في حدث التحميل للنموذج نقوم بالحصول على مقبض نافذة النظام بواسطة الأمر GetSystemMenu ثم نقوم بإضافة البنود التي نريدها لقائمة النظام بواسطة AppendMenu

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal
    Dim hSysMenu As Integer
    hSysMenu = GetSystemMenu(Me.Handle.ToInt32, False)
    'appends new menu items
    AppendMenu(hSysMenu, MF_SEPARATOR, 1000, "")
    AppendMenu(hSysMenu, 0, 1001, "New Item1")
    AppendMenu(hSysMenu, 0, 1002, "New Item2")
End Sub
```

ثالثا – في الـ Override لـ WndProc نستخدم كود شبيه بالتالي لمعرفة أي بند من القائمة تم نقره

```
Protected Overrides Sub WndProc(ByRef m As Message)
    MyBase.WndProc(m)
    If m.Msg = WM_SYSCOMMAND Then
        Select Case m.WParam.ToInt32
            Case 1001
                MsgBox("Clicked: New Item1")
            Case 1002
                MsgBox("Clicked: New Item2")
        End Select
    End If
End Sub
```

كيف يمكننا البحث عن ملف بمحتوى معين ضمن شجرة مجلدات

أنشئ مشروعا جديدا من النوع Windows Forms Application وضع على النافذة TextBox عدد 2 و Button عدد 2 و ListBox عدد 1 وإن أحببت المتابعة بتسمياتي قم بتسمية التحكمات السابقة كما يلي txtPath و txtContents و btnSelectFolder و btnSearch و lstResults

الآن أنشئ معالج لحدث النقر على كلا الزرين واجعل كود زر اختيار المجلد كما يلي

```
Private Sub btnSelectFolder_Click() Handles btnSelectFolder.Click
    ' اختيار المجلد
    Dim sf As New FolderBrowserDialog
    If sf.ShowDialog = Windows.Forms.DialogResult.OK Then
        Me.txtPath.Text = sf.SelectedPath
    End If
End Sub
```

من أجل الاستعلام عن الملفات المطلوبة سنحتاج إلى وظيفة تعيد لنا قائمة بالملفات الموجودة في شجرة مجلدات معينة حيث سنقوم بتمرير مسار المجلد الذي سنبدأ منه كمحدد وحيد لها وسيكون كودها على الشكل التالي حيث استخدمنا استعلام لينك يعيد لنا مجموعة من النوع FileInfo لقائمة الملفات الموجودة في شجرة المجلدات الممررة مستخدما الدالة GetFiles للحصول على قائمة الملفات المطلوبة

```
' إعادة قائمة بالملفات ضمن شجرة مجلدات معينة
Function GetFiles(ByVal root As String) As _
         System.Collections.Generic.IEnumerable(Of System.IO.FileInfo)

    Return From file In My.Computer.FileSystem.GetFiles _
                (root, FileIO.SearchOption.SearchAllSubDirectories, "*.*") _
                Select New System.IO.FileInfo(file)
End Function
```

وسنحتاج وظيفة أخرى تعيد لنا محتويات الملف الذي نقرأه عند الاستعلام حيث نمرر لها مسار ذلك الملف وسيكون كود هذه الوظيفة على الشكل التالي حيث استخدمنا الدالة ReadAllText لقراءة كافة محتويات الملف وإعادتها

```
' قراءة محتويات الملف
Function GetFileText(ByVal FileName As String) As String
    ' إن كان الملف موجودا أعد محتوياته وإلا أعد نصا فارغا
    If System.IO.File.Exists(FileName) Then
        Return System.IO.File.ReadAllText(FileName)
    Else
        Return String.Empty
    End If
End Function
```

وسيكون كود زر البحث كالتالي

```
Private Sub btnSearch_Click() Handles btnSearch.Click
    ' التأكد من المسار وشرط البحث قبل المتابعة
    If Me.txtPath.Text <> String.Empty _
       And Directory.Exists(Me.txtPath.Text) _
       And Me.txtContents.Text <> String.Empty Then
        ' الحصول على قائمة بالملفات
        Dim fileList = GetFiles(Me.txtPath.Text)

        ' استخدام استعلام لينك للحصول على قائمة بالملفات التي تطابق شرط البحث
        Dim queryMatchingFiles = From file In fileList _
                    Where file.Extension = ".htm" _
```

```
                        Let fileText = GetFileText(file.FullName) _
                        Where
fileText.ToUpper.Contains(Me.txtContents.Text.ToUpper) _
                        Select file.FullName

            ' إظهار النتائج في صندوق القائمة
            Me.lstResults.Items.Clear()
            Me.lstResults.Items.AddRange(queryMatchingFiles.ToArray)

        End If

        MsgBox("Done.")
End Sub
```

حيث استخدمنا في البداية عبارة If للتحقق من نص البحث والمسار المطلوب قبل تنفيذ عملية البحث ثم قمنا بالحصول على قائمة الملفات ووضعها في المتغير fileList ثم استخدمنا استعلام لينك يستعلم من fileList وفي قسم Where حددنا أننا نريد البحث في الملفات التي تمتلك اللاحقة htm ثم أضفنا شرطا آخر لقسم Where بأننا نريد الحصول على الملفات التي تحتوي نصا معينا فقط وقمنا باستخدام الدالة ToUpper مع كلا النصين من أجل تجاهل حالة النص (حروف كبيرة أو صغيرة) ثم قمنا بإظهار نتائج الاستعلام في صندوق القائمة

فكرة لتطوير المشروع ولكن لن أنفذها أنا وأنتظر من أحد متابعي موضوعي تنفيذها وهي أنه إذا كانت شجرة المجلدات المبحوث فيها كبيرة فالبحث سيأخذ وقتا طويلا لذا يجب نقل تنفيذ الاستعلام إلى مسار آخر Another Thread وإظهار مؤشر بتقدم العملية باستخدام ProgressBar أسفل النافذة كما يمكن إضافة صندوق نصوص آخر على النافذة لجعل مستخدم تطبيقنا يحدد لاحقة الملفات التي يريد البحث فيها

كيف نقوم بالبحث في سجل النظام

نحتاج أحيانا للبحث في سجل النظام عن مفتاح يحتوي على قيمة أو محتويات محددة ولهذا الغرض نستخدم الفئة Microsoft.Win32.Registry للحصول على الكائن Microsoft.Win32.RegistryKey الذي يمثل المفتاح الجذري للعش الذي نود البحث فيه في سجل النظام مستخدمين عناصر الكائن RegistryKey للتنقل عبر شجرية مفتاح السجل وتعداد عناصره وقراءة أسماء وقيم المفاتيح المحتواة في ذلك المفتاح.

وهنا يجب علينا أولا الحصول على كائن RegistryKey يمثل المستوى الأساسي للتنقل عبر عناصر شجرية الكائن RegistryKey حيث تقدم لنا الفئة Registry مجموعة مؤلفة من سبعة خاصيات مشتركة تمثل المستوى الأساسي لمفاتيح سجل النظام وهي

- ClassesRoot HKEY_CLASSES_ROOT
- CurrentConfig HKEY_CURRENT_CONFIG
- CurrentUser HKEY_CURRENT_USER
- DynData HKEY_DYN_DATA
- LocalMachine HKEY_LOCAL_MACHINE
- PerformanceData HKEY_PERFORMANCE_DATA
- Users HKEY_USERS

كما يوفر لنا الكائن My الفئة My.Computer.Registry التي تتضمن مجموعة مطابقة من الخصائص توفر نفس وظيفية الخصائص الموجودة في الفئة Microsoft.Win32.Registry وبعد حصولك على الكائن RegistryKey الجذري يمكنك التنقل عبر مفاتيحه الفرعية ولدعم هذا التنقل تمكنك الفئة RegistryKey من استخدام الطريقة GetSubKeyNames للحصول على مصفوفة نصية تحتوي على أسماء جميع المفاتيح الفرعية ثم استخدام الطريقة OpenSubKey للحصول على مرجع للمفتاح الفرعي وهي متوفرة بشكلان محملان الأول يفتح ذلك المفتاح للقراءة فقط والثاني يستقبل قيمة منطقية إن كانت True فهي تفتح ذلك المفتاح مع قابلية الكتابة أيضا.

وحالما تحصل على الكائن RegistryKey يمكنك عندها إنشاء أو قراءة أو تحديث أو حتى حذف المفاتيح والقيم الفرعية باستخدام الطرائق التالية مع ملاحظة أنه عند استخدام الطرائق التي تقوم بالتعديلات على محتويات المفتاح يجب أن تكون قد حصلت على كائن RegistryKey قابل للكتابة حتى تستطيع القيام بالتغييرات المطلوبة

- CreateSubKey تنشئ مفتاح فرعي باسم معين وتعيد كائن RegistryKey قابل للكتابة فإن كان ذلك المفتاح موجودا فهي تعيد مرجعا له
- DeleteSubKey تقوم بحذف مفتاح باسم محدد حيث يجب أن لا يحتوي ذلك المفتاح على أية مفاتيح فرعية ويمكن ان يحتوي على قيم وفي حالة فشله فهو يطلق استثناء System.InvalidOperationException
- DeleteSubKeyTree يقوم بحذف المفتاح مع جميع المفاتيح الفرعية والقيم ضمنه
- DeleteValue يقوم بحذف قيمة باسم محدد في المفتاح الحالي
- GetValue يعيد قيمة باسم محدد في المفتاح الحالي وتكون القيمة المعادة من النوع Object ثم يجب استخدام دوال تحويل الأنواع لتحويلها إلى نوع البيانات المطلوب وهي تعيد Nothing إن لم يتم إيجاد القيمة وهي تمتلك طريقة محملة تمكنك من تحديد القيمة الافتراضية المعادة بدلا عن Nothing
- GetValueKind تعيد نوع البيانات الذي تحمله قيمة محددة في المفتاح الحالي وتكون القيمة المعادة من نوع التعداد Microsoft.Win32.RegistryValueKind
- GetValueNames وهي تعيد مصفوفة نصية تحتوي على أسماء جميع القيم المحتواة في المفتاح الحالي وإن كان المفتاح يحتوي على قيمة افتراضية ممثلة بسلسلة نصية فارغة فيتم إعادة هذه السلسلة النصية الفارغة كعنصر ضمن المصفوفة المعادة

- SetValue تقوم بإنشاء أو تعديل قيمة باسم محدد وهي تزودنا بطريقة محملة تمكننا من تحديد نوع البيانات المخزنة في تلك القيمة حيث تأخذ قيمة التعداد RegistryValueKind كمحدد أخير لتحديد نوع تلك القيمة وإن لم نقم بتحديد نوع هذه القيمة فيتم الاستدلال على نوعها آليا اعتمادا على نوع الكائن الممرر لضبط تلك القيمة

كما أن الفئة RegistryKey تحقق الواجهة IDisposable لهذا عليك استدعاء الطريقة IDispsable.Dispose لتحرير مصادر النظام عندما تنتهي من استخدام الكائن RegistryKey.

والمثال التالي يأخذ محدد سطر أوامر وحيد يقوم بالبحث عنه في العش CurrentUser باحثا عن المفاتيح التي يطابق اسمها المحدد الممرر وعندما يجد مفاتيحا مطابقة يقوم بإظهار جميع القيم النصية المحتواة فيه على شاشة الكونسول

```
Imports System
Imports Microsoft.Win32
Namespace Apress.VisualBasicRecipes.Chapter15

  Public Class Recipe15_05

    Public Shared Sub SearchSubKeys(ByVal root As RegistryKey, ↪
                    ByVal searchKey As String)
      ' Loop through all subkeys contained in the current key.
      For Each keyName As String In root.GetSubKeyNames
        Try
          Using key As RegistryKey = root.OpenSubKey(keyName)
            If keyName = searchKey Then PrintKeyValues(key)
              SearchSubKeys(key, searchKey)
          End Using
          Catch ex As Security.SecurityException
            ' Ignore SecurityException for the purpose of this example.
            ' Some subkeys of HKEY_CURRENT_USER are secured and will
            ' throw a SecurityException when opened.
          End Try
        Next
      End Sub

      Public Shared Sub PrintKeyValues(ByVal key As RegistryKey)
        ' Display the name of the matching subkey and the number of
        ' values it contains.
        Console.WriteLine("Registry key found : {0} contains {1} values", ↪
            key.Name, key.ValueCount)
        ' Loop through the values and display.
        For Each valueName As String In key.GetValueNames
            If TypeOf key.GetValue(valueName) Is String Then
              Console.WriteLine(" Value : {0} = {1}", valueName, ↪
                    key.GetValue(valueName))
            End If
        Next
      End Sub

      Public Shared Sub Main(ByVal args As String())
        If args.Length > 0 Then
          ' Open the CurrentUser base key.
          Using root As RegistryKey = Registry.CurrentUser
            ' Search recursively through the registry for any keys
            ' with the specified name.
            SearchSubKeys(root, args(0))
          End Using
```

```
            End If
            ' Wait to continue.
            Console.WriteLine(Environment.NewLine)
            Console.WriteLine("Main method complete. Press Enter.")
            Console.ReadLine()
        End Sub

    End Class
End Namespace
```

و بتشغيل المثال السابق واستخدام Environment كمحدد سطر أوامر فسوف يظهر لنا خرجا شبيها بالتالي إن كنا نعمل على جهاز يشغل ويندوز فيستا

```
Registry key found : HKEY_CURRENT_USER\Environment contains 3 values
Value : TEMP = C:\Users\ Todd \AppData\Local\Temp
Value : TMP = C:\Users\Todd\AppData\Local\Temp
...
Main method complete. Press Enter.
```

تعالوا نعمل معا Task Manager بسيط وبسرعة

- أي نسخة من فيجول ستوديو 2008 ستعمل معنا حتى الـ Express
- يعتبر هذا البرنامج مثالا عمليا على Linq To Object حيث نستخدم استعلامات Linq للحصول على المعلومات المطلوبة

- أنشئ مشروعا جديدا وسمه ما تشاء ثم ضع على النموذج DataGridView وأبق على الاسم الافتراضي DataGridView1 إذا أحببت المتابعة بتسمياتي ثم ضع زرين أسفل الـ DataGridView وأعط لأحدهما اسما btnFill واجعل الخاصية Text مساوية لـ Fill و الآخر سمه btnKill واجعل الخاصية Text مساوية لـ Kill
- افتح خصائص MyProject و من الصفحة Application اضغط زر view UAC settings وفي محرر الكود الذي يظهر لك استبدل السطر

```
<requestedExecutionLevel  level="asInvoker" uiAccess="false" />
```

بالسطر

```
<requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
```

في محرر الكود للنموذج Form1 أدخل الكود التالي الذي سيكون هو الكود الكامل لبرنامجنا

```
Public Class Form1

    Private Bs As New BindingSource
    Private Sub btnFill_Click() Handles btnFill.Click
        Me.DataGridView1.DataSource = Bs
        Dim pr = From a In Process.GetProcesses _
                 Order By a.ProcessName _
                 Select a.Id, a.ProcessName, a.MainWindowTitle, _
                 HandlesCount = a.HandleCount, ThreadsCount = a.Threads.Count

        Me.Bs.DataSource = pr
    End Sub

    Private Sub btnKill_Click() Handles btnKill.Click
        Dim pra = Bs.Current
        Dim prk = From a In Process.GetProcesses _
                  Where a.Id = pra.Id _
                  And a.MainWindowTitle = pra.MainWindowTitle _
                  Select a

        For Each p In prk
            p.Kill()
        Next

        btnFill_Click()
    End Sub

End Class
```

في البداية عرفنا متغيرا Bs من النوع BindingSource الذي سنستخدمه لربط نتيجة الاستعلام مع DataGridView ثم في كود الزر Fill قمنا بربط Bs مع DatagridView ثم كتبنا استعلام Linq ليجلب لنا الحقول التي نريدها من ناتج الطريقة Process.GetProcesses وقمنا بالترتيب حسب ProcessName ثم اخترنا الحقول التي نرغب بإظهارها في شبكة البيانات ثم قمنا بضبط قيمة الخاصية DataSource إلى استعلامنا Pr مما سيسبب إظهار نتيجة الاستعلام في شبكة البيانات

وفي كود الزر Kill جلبنا أولا قيمة السطر الذي يقف عنده المؤشر في شبكة البيانات بالحصول على قيمة الخاصية Current لـ Bs ثم كتبنا استعلاما شبيها بالسابق يجلب لنا العمليات التي توافق شرطنا بحيث يكون ID و MainWindowTitle مطابقان ثم قمنا بالدوران خلال نتائج الاستعلام عبر حلقة For ... Each واستخدمنا الطريقة kill لإنهاء العمليات التي يعيدها الاستعلام

إذا أردنا إضافة زر إخفاء وإظهار للنافذة الرئيسية للبرنامج - العملية - أضف زرين للنموذج باسم btnHide و btnShow ثم في قسم التصريحات العامة ضمن الفئة Form1 أضف التصريحات التالية

```
Public Declare Auto Function ShowWindow Lib "user32" _
                (ByVal hwnd As Integer, ByVal nCmdShow As Integer) As Integer

Public Const SW_HIDE = 0
Public Const SW_SHOW = 5
Public Const SW_SHOWNA = 8
Public Const SW_SHOWNORMAL = 1
```

ويكون كود زري الإظهار والإخفاء

```
Private Sub btnHide_Click() Handles btnHide.Click
        Dim pra = Bs.Current
        Dim prk = From a In Process.GetProcesses _
                  Where a.Id = pra.Id _
                  And a.MainWindowTitle = pra.MainWindowTitle _
                  Select a

        For Each p In prk
            ShowWindow(p.MainWindowHandle, SW_HIDE)
        Next

        btnFill_Click()
    End Sub

Private Sub btnShow_Click() Handles btnShow.Click
        Dim pra = Bs.Current
        Dim prk = From a In Process.GetProcesses _
                  Where a.Id = pra.Id _
                  And a.MainWindowTitle = pra.MainWindowTitle _
                  Select a

        For Each p In prk
            ShowWindow(p.MainWindowHandle, SW_SHOW)
        Next

        btnFill_Click()
    End Sub
```

ومن أجل نقل التركيز لتطبيق معين نحتاج إلى التعريفات التالية

```
Public Declare Auto Function SetForegroundWindow Lib "user32" _
                        (ByVal hwnd As Integer) As Boolean
```

ونضيف زر باسم SetFocus يكون كوده كما يلي

```
Private Sub btnSetfocus_Click() Handles btnSetfocus.Click
    Dim pra = Bs.Current
    Dim prk = From a In Process.GetProcesses _
              Where a.Id = pra.Id _
              And a.MainWindowTitle = pra.MainWindowTitle _
              Select a

    For Each p In prk
        SetForegroundWindow(p.MainWindowHandle)
    Next

    btnFill_Click()
End Sub
```

تعقب إضافة وإزالة الأقراص المرتبطة عبر منفذ Usb

فكرة العمل: بناء فئة Class للتعامل مع الأقراص القابلة للإزالة والتي يتم توصيلها إلى الحاسب عبر منفذ USB

الفوائد: مثال على البرمجة غرضية التوجه، بناء الفئات، إضافة الخصائص، إطلاق الأحداث

سنقوم أولا بتعريف فئة Class لتضم عملنا وعندما ننتهي منها يجب أن تضم العديد من الخصائص والإجراءات المفيدة

```
Public Class UsbDriveDetect

End Class
```

سنحتاج لإضافة مرجع لـ ManagementEventWatcher حتى نستطيع مراقبة ما يحدث داخل الجهاز والذي يقوم بمراقبة الأحداث المنطلقة بناء على استعلام معين و سنستخدم الكلمة WithEvents في التعريف حتى نستطيع تعقب الأحداث الصادرة عنه

```
Private WithEvents m_MediaConnectWatcher As ManagementEventWatcher
```

ولكي يعمل بشكل صحيح سنحتاج لإضافة مرجع لـ System.Management الآن قم بإضافة System.Management من قائمة Project بند Add Reference ثم استخدم الاستيراد التالي قبل كل شئ في الملف وحتى قبل تعريف الفئة أيضا

```
Imports System.Management
```

سنحتاج أيضا لبناء مشيد الفئة New Sub الذي سيضبط التهيئة الأساسية لفئتنا والذي سنقوم من خلاله بضبط خصائص ManagementEventWatcher الذي يقوم بالعمل لأجلنا

```
Sub New()
     Dim Query2 As New WqlEventQuery("SELECT * FROM __InstanceOperationEvent WITHIN 1 " _
               & "WHERE TargetInstance ISA 'Win32_DiskDrive'")

        Me.m_MediaConnectWatcher = New ManagementEventWatcher
        Me.m_MediaConnectWatcher.Query = Query2
End Sub
```

سنقوم الآن بوضع إجراءات لبدء وإيقاف تعقب وضع وإزالة أقراص USB

```
Public Sub StartDetection()
        Me.m_MediaConnectWatcher.Start()
End Sub

    Public Sub StopDetection()
        Me.m_MediaConnectWatcher.Stop()
End Sub
```

نريد الآن إعلام مستخدم فئتنا عن طريق إطلاق حدث خاص عندما يتم وضع أو إزالة قرص عن طريق منفذ USB في الجهاز سنقوم بإنشاء فئة مشتقة من System.EventArgs و ذلك كفئة فرعية ضمن فئتنا الأساسية UsbDriveDetect حتى نستخدمها لإطلاق حدثنا ولكن سنحتاج أولا للتعريف التالي لإعادة حالة القرص هل تم وضعه أو إزالته ضع التعريف التالي بعد بداية تعريف الفئة

```
Public Enum EnUsbArrivedRemoved
```

```
        Arrived
        Removed
End Enum
```

وفيما يلي نص الفئة الخاصة بالحدث

```
Public Class UsbDriveEeventArgs
        Inherits System.EventArgs

        Private m_DeviceName As String
        Private m_DriveLetter As String
        Private m_ArrivedRemoved As EnUsbArrivedRemoved

        Sub New(ByVal DeviceName As String, ByVal DriveLetter As String, _
                   ByVal ArrivedRemoved As EnUsbArrivedRemoved)

            Me.m_DeviceName = DeviceName
            Me.m_DriveLetter = DriveLetter
            Me.m_ArrivedRemoved = ArrivedRemoved
        End Sub

        Public ReadOnly Property DeviceName() As String
            Get
                Return Me.m_DeviceName
            End Get
        End Property

        Public ReadOnly Property DriveLetter() As String
            Get
                Return Me.m_DriveLetter
            End Get
        End Property

        Public ReadOnly Property ArrivedRemoved() As EnUsbArrivedRemoved
            Get
                Return Me.m_ArrivedRemoved
            End Get
        End Property

    End Class
```

و أيضا بعد بداية تعريف الفئة ضع السطر التالي الذي سيعرف الحدث الذي سنقوم بإطلاقه

```
Public Event UsbDeviceArrivedRemoved(ByVal sender As Object, _
                         ByVal e As UsbDriveEeventArgs)
```

قبل استخدام الحدث UsbDeviceArrivedRemoved لإعلام المستخدم بوضع أو إزالة قرص USB يجب الحصول على حرف ذلك القرص

```
Private Function GetDriveLetterFromDisk(ByVal Name As String) As String
        Dim oq_part, oq_disk As ObjectQuery
        Dim mos_part, mos_disk As ManagementObjectSearcher
        Dim obj_part, obj_disk As ManagementObject
        Dim ans As String = ""

        ' WMI queries use the "\" as an escape charcter
        Name = Replace(Name, "\", "\\")
```

```
        ' First we map the Win32_DiskDrive instance with the association called
        ' Win32_DiskDriveToDiskPartition. Then we map the Win23_DiskPartion
        ' instance with the assocation called Win32_LogicalDiskToPartition

       oq_part = New ObjectQuery("ASSOCIATORS OF {Win32_DiskDrive.DeviceID=""" _
                       & Name & """} WHERE AssocClass = _
                       Win32_DiskDriveToDiskPartition")

        mos_part = New ManagementObjectSearcher(oq_part)
        For Each obj_part In mos_part.Get()

            oq_disk = New ObjectQuery("ASSOCIATORS OF " _
                 & "{Win32_DiskPartition.DeviceID=""" & obj_part("DeviceID") _
                 & """} WHERE AssocClass = Win32_LogicalDiskToPartition")

            mos_disk = New ManagementObjectSearcher(oq_disk)
            For Each obj_disk In mos_disk.Get()
                ans &= obj_disk("Name") & ","
            Next
        Next

        Return ans.Trim(","c)
End Function
```

حيث نمرر له اسم الجهاز الذي تم الكشف عن إضافته وهو يعيد لنا حرف السواقة المرتبطة به وذلك باستخد ام ManagementObjectSearcher للحصول عليه حيث ستلاحظ أن طريقة الاستعلام هنا مشابهة لاستعلامات قواعد البيانات ولكنها هنا على فئات WMI بدلا من جداول قاعدة البيانات

```
Private Function GetDriveLetterFromDisk(ByVal Name As String) As String
        Dim oq_part, oq_disk As ObjectQuery
        Dim mos_part, mos_disk As ManagementObjectSearcher
        Dim obj_part, obj_disk As ManagementObject
        Dim ans As String = ""

        ' WMI queries use the "\" as an escape charcter
        Name = Replace(Name, "\", "\\")

        ' First we map the Win32_DiskDrive instance with the association called
        ' Win32_DiskDriveToDiskPartition. Then we map the Win23_DiskPartion
        ' instance with the assocation called Win32_LogicalDiskToPartition

       oq_part = New ObjectQuery("ASSOCIATORS OF {Win32_DiskDrive.DeviceID=""" _
                & Name & """} WHERE AssocClass = _
                Win32_DiskDriveToDiskPartition")

        mos_part = New ManagementObjectSearcher(oq_part)
        For Each obj_part In mos_part.Get()

            oq_disk = New ObjectQuery("ASSOCIATORS OF _
               {Win32_DiskPartition.DeviceID=""" & obj_part("DeviceID") _
               & """} WHERE AssocClass = Win32_LogicalDiskToPartition")

            mos_disk = New ManagementObjectSearcher(oq_disk)
            For Each obj_disk In mos_disk.Get()
```

```
                ans &= obj_disk("Name") & ","
            Next
        Next

        Return ans.Trim(","c)
End Function
```

والآن سنتتبع الحدث EventArrived الخاص بـ ManagementEventWatcher حتى نعرف متى تم وضع أو إزالة قرص USB وذلك كي نستطيع إطلاق حدثنا المناسب وفقا للحدث المستقبل

```
Private Sub m_MediaConnectWatcher_EventArrived(ByVal sender As Object, _
             ByVal e As System.Management.EventArrivedEventArgs) Handles _
               m_MediaConnectWatcher.EventArrived

        Dim mbo, obj As ManagementBaseObject
        ' the first thing we have to do is figure out if this is
        ' a creation or deletion event

        mbo = CType(e.NewEvent, ManagementBaseObject)
        ' next we need a copy of the instance that was either created or deleted
        obj = CType(mbo("TargetInstance"), ManagementBaseObject)
        If obj("InterfaceType") = "USB" Then
            Select Case mbo.ClassPath.ClassName
                Case "__InstanceCreationEvent"
                    Dim Ee As New UsbDriveEeventArgs(obj("Caption"), _
               GetDriveLetterFromDisk(obj("Name")), EnUsbArrivedRemoved.Arrived)

                    RaiseEvent UsbDeviceArrivedRemoved(Me, Ee)
                Case "__InstanceDeletionEvent"
                    Dim Ee As New UsbDriveEeventArgs(obj("Caption"), "", _
                    EnUsbArrivedRemoved.Removed)

                    RaiseEvent UsbDeviceArrivedRemoved(Me, Ee)
            End Select
        End If
End Sub
```

سوف أقوم بإضافة متغير يعيد بعض المعلومات عن ذلك القرص ولذلك سنحتاج إلى تعريف تركيب Structure ليعيد تلك المعلومات

```
    Public Structure DriveInfoStr
        Dim DriveLetter As String
        Dim Description As String
        Dim FileSystem As String
        Dim Size As UInt64
        Dim FreeSpace As UInt64
        Dim DriveType As DriveTypeEnum
        Dim VolumeName As String
        Dim VolumeSerialNumber As String
    End Structure
```

وأيضا إلى تعداد Enum ليعيد نوع القرص

```
    Public Enum DriveTypeEnum
        Unknown = 0
        NoRootDirectory = 1
        RemovableDisk = 2
```

```
        LocalDisk = 3
        NetworkDrive = 4
        CompactDisc = 5
        RAMDisk = 6
    End Enum
```

وسنضيف ذلك إلى UsbDriveEeventArgs حتى تعيد القيمة

```
Private m_DriveInfoS As DriveInfoStr

Public ReadOnly Property DriveInformation() As DriveInfoStr
    Get
        Return m_DriveInfoS
    End Get
End Property
```

وسنعدل Sub New أيضا لـ UsbDriveEeventArgs

```
Sub New(ByVal DeviceName As String, ByVal DriveLetter As String, _
        ByVal ArrivedRemoved As EnUsbArrivedRemoved _
        , ByVal DriveInformations As DriveInfoStr)

     Me.m_DeviceName = DeviceName
     Me.m_DriveLetter = DriveLetter
     Me.m_ArrivedRemoved = ArrivedRemoved
     Me.m_DriveInfoS = DriveInformations
 End Sub
```

وفيما يلي إجراء الحصول على المعلومات

```
Private Function GetDriveInformation(ByVal DriveLetter As String) As
DriveInfoStr

    Dim Query As String = "Select * from Win32_LogicalDisk WHERE DeviceID = '" _
                       & DriveLetter & "'"

    Dim colDisks As New ManagementObjectSearcher(Query)
    For Each objDisk As ManagementObject In colDisks.Get
        Dim DrIn As DriveInfoStr
        With DrIn
            .DriveLetter = objDisk("DeviceID")
            .FileSystem = objDisk("FileSystem")
            .Size = objDisk("Size")
            .FreeSpace = objDisk("FreeSpace")
            .Description = objDisk("Description")
            .DriveType = objDisk("DriveType")
            .VolumeName = objDisk("VolumeName")
            .VolumeSerialNumber = objDisk("VolumeSerialNumber")
        End With
        Return DrIn
    Next
    Return Nothing
End Function
```

شئ أخير سنقوم بتغييره وهو كود إطلاق الحدث ليتوافق مع الإضافات الجديدة

```
Dim DrL As String = GetDriveLetterFromDisk(obj("Name"))
Dim Ee As New UsbDriveEeventArgs(obj("Caption"), DrL, _
             EnUsbArrivedRemoved.Arrived, GetDriveInformation(DrL))
```

```
RaiseEvent UsbDeviceArrivedRemoved(Me, Ee)
```

وفيما يلي نص الفئة الكامل

```
Public Class UsbDriveDetect

    Private WithEvents m_MediaConnectWatcher As ManagementEventWatcher

    Public Enum EnUsbArrivedRemoved
        Arrived
        Removed
    End Enum

    Public Event UsbDeviceArrivedRemoved(ByVal sender As Object, ByVal e As
UsbDriveEeventArgs)

    Public Structure DriveInfoStr
        Dim DriveLetter As String
        Dim Description As String
        Dim FileSystem As String
        Dim Size As UInt64
        Dim FreeSpace As UInt64
        Dim DriveType As DriveTypeEnum
        Dim VolumeName As String
        Dim VolumeSerialNumber As String
    End Structure

    Public Enum DriveTypeEnum
        Unknown = 0
        NoRootDirectory = 1
        RemovableDisk = 2
        LocalDisk = 3
        NetworkDrive = 4
        CompactDisc = 5
        RAMDisk = 6
    End Enum

    Sub New()
      Dim Query2 As New WqlEventQuery("SELECT * FROM __InstanceOperationEvent" _
            & " WITHIN 1 WHERE TargetInstance ISA 'Win32_DiskDrive'")

        Me.m_MediaConnectWatcher = New ManagementEventWatcher
        Me.m_MediaConnectWatcher.Query = Query2
    End Sub

    Public Sub StartDetection()
        Me.m_MediaConnectWatcher.Start()
    End Sub

    Public Sub StopDetection()
        Me.m_MediaConnectWatcher.Stop()
    End Sub

    Public Class UsbDriveEeventArgs
        Inherits System.EventArgs
```

```
    Private m_DeviceName As String
    Private m_DriveLetter As String
    Private m_ArrivedRemoved As EnUsbArrivedRemoved
    Private m_DriveInfoS As DriveInfoStr

    Sub New(ByVal DeviceName As String, ByVal DriveLetter As String, _
            ByVal ArrivedRemoved As EnUsbArrivedRemoved _
            , ByVal DriveInformations As DriveInfoStr)
        Me.m_DeviceName = DeviceName
        Me.m_DriveLetter = DriveLetter
        Me.m_ArrivedRemoved = ArrivedRemoved
        Me.m_DriveInfoS = DriveInformations
    End Sub

    Public ReadOnly Property DriveInformation() As DriveInfoStr
        Get
            Return m_DriveInfoS
        End Get
    End Property

    Public ReadOnly Property DeviceName() As String
        Get
            Return Me.m_DeviceName
        End Get
    End Property

    Public ReadOnly Property DriveLetter() As String
        Get
            Return Me.m_DriveLetter
        End Get
    End Property

    Public ReadOnly Property ArrivedRemoved() As EnUsbArrivedRemoved
        Get
            Return Me.m_ArrivedRemoved
        End Get
    End Property

End Class

Private Sub m_MediaConnectWatcher_EventArrived(ByVal sender As Object, _
           ByVal e As System.Management.EventArrivedEventArgs) Handles _
           m_MediaConnectWatcher.EventArrived

    Dim mbo, obj As ManagementBaseObject
    ' the first thing we have to do is figure out if this is
    ' a creation or deletion event
    mbo = CType(e.NewEvent, ManagementBaseObject)
    ' next we need a copy of the instance that was either created or deleted
    obj = CType(mbo("TargetInstance"), ManagementBaseObject)
    If obj("InterfaceType") = "USB" Then
        Select Case mbo.ClassPath.ClassName
            Case "__InstanceCreationEvent"
                Dim DrL As String = GetDriveLetterFromDisk(obj("Name"))
                Dim Ee As New UsbDriveEeventArgs(obj("Caption"), _
```

```
                DrL, EnUsbArrivedRemoved.Arrived, GetDriveInformation(DrL))

                RaiseEvent UsbDeviceArrivedRemoved(Me, Ee)
            Case "__InstanceDeletionEvent"
                Dim Ee As New UsbDriveEeventArgs(obj("Caption"), Nothing, _
                         EnUsbArrivedRemoved.Removed, Nothing)

                RaiseEvent UsbDeviceArrivedRemoved(Me, Ee)
        End Select
    End If
End Sub

Private Function GetDriveLetterFromDisk(ByVal Name As String) As String
    Dim oq_part, oq_disk As ObjectQuery
    Dim mos_part, mos_disk As ManagementObjectSearcher
    Dim obj_part, obj_disk As ManagementObject
    Dim ans As String = ""

    ' WMI queries use the "\" as an escape charcter
    Name = Replace(Name, "\", "\\")

    ' First we map the Win32_DiskDrive instance with the association called
    ' Win32_DiskDriveToDiskPartition. Then we map the Win23_DiskPartion
    ' instance with the assocation called Win32_LogicalDiskToPartition

   oq_part = New ObjectQuery("ASSOCIATORS OF {Win32_DiskDrive.DeviceID=""" _
          & Name & """} WHERE AssocClass = Win32_DiskDriveToDiskPartition")

    mos_part = New ManagementObjectSearcher(oq_part)
    For Each obj_part In mos_part.Get()

        oq_disk = New ObjectQuery("ASSOCIATORS OF " & _
        {Win32_DiskPartition.DeviceID=""" & obj_part("DeviceID") _
         & """} WHERE AssocClass = Win32_LogicalDiskToPartition")

        mos_disk = New ManagementObjectSearcher(oq_disk)
        For Each obj_disk In mos_disk.Get()
            ans &= obj_disk("Name") & ","
        Next
    Next

    Return ans.Trim(","c)

End Function

Private Function GetDriveInformation(ByVal DriveLetter As String) _
                      As DriveInfoStr

    Dim Query As String = "Select * from Win32_LogicalDisk WHERE " & _
                       DeviceID = '" & DriveLetter & "'"

    Dim colDisks As New ManagementObjectSearcher(Query)
    For Each objDisk As ManagementObject In colDisks.Get
        Dim DrIn As DriveInfoStr
        With DrIn
```

```
                .DriveLetter = objDisk("DeviceID")
                .FileSystem = objDisk("FileSystem")
                .Size = objDisk("Size")
                .FreeSpace = objDisk("FreeSpace")
                .Description = objDisk("Description")
                .DriveType = objDisk("DriveType")
                .VolumeName = objDisk("VolumeName")
                .VolumeSerialNumber = objDisk("VolumeSerialNumber")
            End With
            Return DrIn
        Next
        Return Nothing
    End Function

End Class
```

وفيما يلي مثال عن الاستخدام - عرف متغيرا عاما يشير إلى فئتنا ضمن فئة النموذج

```
Private WithEvents UsbMonitor As New UsbDriveDetect
```

أضف زري أوامر لبدء وإيقاف التعقب

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles Button1.Click

        Me.UsbMonitor.StartDetection()
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button2.Click

        Me.UsbMonitor.StopDetection()
End Sub
```

عالج الحدث الصادر عن فئتنا إظهار رسالة مثلا

```
Private Sub UsbMonitor_UsbDeviceArrivedRemoved(ByVal sender As Object, _
                   ByVal e As UsbDriveDetect.UsbDriveEeventArgs) Handles

    UsbMonitor.UsbDeviceArrivedRemoved
    MsgBox(e.DeviceName & ControlChars.CrLf & e.DriveLetter _
            & ControlChars.CrLf & e.ArrivedRemoved.ToString)
End Sub
```

تشغيل برنامج خارجي من ضمن كود فيجول بايزيك دوت نيت

مقدمة

كنا أيام VB6 نستخدم الأمر Shell لتشغيل برنامج خارجي فمثلا لتشغيل الآلة الحاسبة كنا نستخدم الكود

```
Shell("calc.exe")
```

ولكن مع تطور لغة البايزك ومع قدوم الدوت نيت تم تقديم طريقة جديدة لتشغيل برنامج خارجي عن طريق Process.Start حيث يمكننا تشغيل الآلة الحاسبة باستخدام الوظيفة الجديدة وبكود مكافئ للكود السابق

```
Process.Start("calc.exe")
```

إضافة إلى أن الأمر Shell موجود ضمن مجال الأسماء Microsoft.VisualBasic الموجود بغرض التوافقية المرجعية مع VB6 ولاينصح باستخدام الأوامر الموجودة في مجال الأسماء المذكور وذلك لتوفر بدائل أفضل وأكثر قوة عن تلك الموجودة فيه إضافة إلى أننا لانضمن متى تقوم مايكروسوفت بإلغاء دعم التوافقية مع VB6

ولكن ماذا لو احتجنا لتنفيذ سطر أوامر طويل أو معقد بالطبع ستقولون لي بأنه من الصعب تنفيذه باستخدام أي من الكودين السابقين لهذا أتت الوظيفة Process.Start بعدة أشكال وذلك لتلبية متطلبات المبرمج المختلفة وفيما يلي بعض الأمثلة عن الوظيفة Start

```
' Start Internet Explorer. Defaults to the home page.
Process.Start("IExplore.exe")

' Start a Web page using a browser associated with .html and .asp files.
Process.Start("IExplore.exe", "C:\myPath\myFile.htm")

' Open web site using IE
Process.Start("IExplore.exe", "www.aya.sy")

' OpenWithStartInfo()
Dim startInfo As New ProcessStartInfo("IExplore.exe")
startInfo.WindowStyle = ProcessWindowStyle.Minimized
Process.Start(startInfo)
startInfo.Arguments = "www.northwindtraders.com"
Process.Start(startInfo)
```

نلاحظ من المثال الأول أننا قمنا بتمرير اسم البرنامج فقط للوظيفة حتى يتم تشغيله وفي المثال الثاني نريد تمرير ملف للمتصفح ليفتحه فقمنا بتمرير اسم الملف التنفيذي لمتصفح الانترنت في المحدد الأول للوظيفة وفي المحدد الثاني نضع محددات سطر الأوامر الخاصة بالبرنامج الذي نريد تشغيله – متصفح الانترنت - وفي مثالنا هنا اسم الملف الذي نريد فتحه مع مساره الكامل وقد نريد فتح موقع انترنت محدد فعندها نضبط قمية المحدد الثاني إلى عنوان ذلك الموقع كما في المثال الثالث وفي المثال الأخير قمنا بتمرير متغير من النوع ProcessStartInfo للوظيفة Start كمحدد وحيد وذلك بعد ضبط الخصائص المناسبة فقد تم تمرير اسم البرنامج كمحدد للباني ثم تم تحديد قيمة الخاصية WindowStyle إلى Minimized وذلك يؤدي إلى تشغيل البرنامج بنافذة مصغرة وتم تحديد قيمة الخاصية Arguments التي تستخدم لتمرير محددات سطر الأوامر إلى التطبيق المراد تشغيله إلى عنوان الموقع الذي نريد فتحه

الفئة ProcessStartInfo

تستخدم هذه الفئة لتحديد خصائص العملية التي نريد تشغيلها وهي تمتلك العديد من الخصائص المفيدة التي تساعدنا على القيام بالمهام المختلفة التي يتطلبها تشغيل عملية – برنامج – ما وفيما يلي استعراض لأهم خصائص هذه الفئة

نبدأ أولا ببباني الفئة الذي يأتي بثلاثة صيغ

ProcessStartInfo()
ProcessStartInfo(String)
ProcessStartInfo(String, String)

فالصيغة الأولى تستخدم لإنشاء متغير من النوع ProcessStartInfo دون أن يتم تمرير أي قيمة لها والثانية يتم تمرير قيمة نصية وحيدة لها هي عبارة عن اسم العملية أو الملف التنفيذي للتطبيق المراد تشغيله والصيغة الأخيرة يتم تمرير محددين نصيين لها الأول عبارة عن اسم العملية أو الملف التنفيذي للتطبيق والمحدد الثاني عبارة عن محددات سطر الأوامر الخاصة بالتطبيق المراد تشغيله. وفيما يلي سرد لأهم خصائص الفئة ProcessStartInfo

- Arguments تحمل قيمة نصية تمثل محددات سطر الأوامر للتطبيق الذي نريد تشغيله
- CreateNoWindow هي قيمة منطقية تشير إلى هل يجب أن يتم تشغيل التطبيق في نافذة جديدة وتكون قيمتها الافتراضية False
- ErrorDialog وهي عبارة عن قيمة منطقية تحدد وجوب إظهار رسالة خطأ في حالة عدم التمكن من تشغيل العملية
- FileName وهي قيمة نصية تحدد اسم ملف البرنامج أو الوثيقة التي سيتم تشغيلها
- LoadUserProfile وهي قيمة منطقية تحدد فيما إذا كان يجب أن يتم تحميل التشكيل الجانبي للمستخدم من سجل النظام
- Password وهي سلسلة نصية تحوي على كلمة السر المستخدمة لبدء العملية
- UserName وهي قيمة نصية تحوي اسم المستخدم المستخدم لبدء العملية
- UseShellExecute وهي قيمة منطقية تحدد فيما إذا كان سيتم استخدام قشرة النظام System Shell لبدء العملية وقيمتها الافتراضية True
- Verb وهي قيمة نصية تحدد العمل الذي سيتم تنفيذه عند بدء العملية والقيمة الافتراضية سلسلة نصية فارغة وفي حالة كونها فارغة يتم تنفيذ العمل الافتراضي المرتبط بذلك الملف
- Verbs وهي عبارة عن مصفوفة قيم نصية تشكل قائمة بالأعمال المرتبطة مع الملف المحدد في الخاصية FileName
- WindowStyle وهي تحمل قيمة ProcessWindowStyle تحدد كيف سيتم إظهار النافذة ويكون لها إحدى القيم Normal أو Hidden أو Minimized أو Maximized
- WorkingDirectory وهي سلسلة نصية تتضمن مسار مجلد العمل الخاص بالتطبيق الذي سيتم تشغيله

ويمكنك الإطلاع على باقي خصائص الفئة ProcessStartInfo من مكتبة MSDN المرفقة مع فيجول ستوديو أو من موقع MSDN الخاص بمايكروسوفت

استخدام ProcessStartInfo مع Process.Start عمليا

لطباعة مستند نصي مثلا إلى الطابعة يمكننا استخدام الكود التالي الذي نحدد فيه الخاصية FileName إلى اسم ملف نصي متواجد في مكان ما على القرص ونحدد الخاصية Verb إلى القيمة Print ثم نستدعي Process.Start

```
Dim Psi As New ProcessStartInfo
With Psi
    .FileName = "D:\Temp\UNTITLED.TXT"
    .Verb = "print"
End With
Process.Start(Psi)
```

ولإظهار الأعمال المرتبطة بهذا الملف

```
For Each ve In Psi.Verbs
    MsgBox(ve.ToString)
Next
```

ولفتح الملف عوضا عن طباعته نغير قيمة الخاصية Verb في الكود السابق إلى Open كما يلي

```
Dim Psi As New ProcessStartInfo
With Psi
    .FileName = "D:\Temp\UNTITLED.TXT"
    .verb = "open"
End With
Process.Start(Psi)
```

ولفتح ملف نصي باستخدام المفكرة وجعل نافذة المفكرة في وضع التكبير نستخدم الكود التالي

```
Dim Psi As New ProcessStartInfo
With Psi
     .FileName = "Notepad.exe"
     .Arguments = "D:\Temp\UNTITLED.TXT"
     .WindowStyle = ProcessWindowStyle.Maximized
End With
```

```
Process.Start(Psi)
```

حيث حددنا اسم الملف التنفيذي للتطبيق – المفكرة هنا – في الخاصية FileName وحددنا اسم الملف الذي نريد فتحه في الخاصية Arguments الخاصة بمحددات سطر الأوامر للتطبيق المذكور اسمه في الخاصية FileName ثم حددنا أن المفكرة يجب أن يتم تشغيلها في وضع تكبير النافذة للحد الأقصى وذلك بضبط الخاصية WindowStyle إلى القيمة ProcessWindowStyle.Maximized ثم قمنا ببدء التطبيق – المفكرة – وذلك باستدعاء الوظيفة Process.Start ممررا لها محددا وحيدا هو Psi الذي عرفناه من النوع ProcessStartInfo

وفيما يلي مثال آخر قد يبدو معقدا قليلا ولكن يمكن شرحه بسهولة باستخدام المعلومات الواردة هنا فقد دار بيني وبين أحد الإخوة نقاش حول بناء الملف التنفيذي للمشروع دون الاعتماد على بيئة التطوير وكان الحل هو باستخدام الأمر MSBuild الذي يأتي مع الفريمورك وهذا الأمر له العديد من محددات بدء التشغيل ونريد أن نقوم بالتنفيذ بشكل مخفي ثم نقوم بإظهار ملف نتيجة عملية البناء بعد إنتهائها وهذا هو الكود مدعما ببعض التعليقات باللغة العربية

```
' جلب مجلد الويندوز
Dim windir = System.Environment.ExpandEnvironmentVariables("%SYSTEMROOT%")

' تحديد متغير معلومات بدء تشغيل العملية
Dim pri As New ProcessStartInfo
With pri
    ' تحديد مجلد العمل
    .WorkingDirectory = "D:\Temp\DisksArchive"
    ' تحديد الأمر الذي نود تنفيذه
    .FileName = windir & "\Microsoft.NET\Framework\v3.5\MSBuild.exe"
    ' تحديد محددات سطر الأوامر من إنشاء ملف بالنتيجة وتحديد اسم المشروع الذي ستتم ترجمته
    .Arguments = "/l:FileLogger,Microsoft.Build.Engine;logfile=ActivityLog.txt DisksArchive.sln"
    ' تحديد أن نافذة التشغيل ستكون مخفية كي لايرى المستخدم نافذة الكونسول
    .WindowStyle = ProcessWindowStyle.Hidden
End With
' بدء تشغيل العملية
Dim pr = Process.Start(pri)
' انتظار العملية حتى يكتمل تنفيذها وتخرج
pr.WaitForExit()
' تغيير المحددات كي نظهر ملف النتائج
With pri
    ' تحديد أن البرنامج الذي سينفذ هو المفكرة
    .FileName = Environment.SystemDirectory & "\notepad.exe"
    ' تحديد اسم الملف الذي نود فتحه
    .Arguments = "ActivityLog.txt"
    ' تحديد أن النافذة يجب أن تظهر بالوضع الطبيعي
    .WindowStyle = ProcessWindowStyle.Normal
End With
' بدء تشغيل العملية
Process.Start(pri)
```

النقطة الوحيدة في هذا الكود الغير مشروحة هنا هو أن الوظيفة Process.Start عند نجاحها في بدء تشغيل العملية تعيد قيمة من النوع System.Diagnostics.Process حيث نضع هذه القيمة في متغير يمكننا من الاستفادة من خصائص الفئة Process المعادة وهنا استخدمنا الوظيفة WaitForExit التي توقف تنفيذ الكود حتى تنتهي العملية من التنفيذ وتخرج وبعدها يتم معاودة تنفيذ باقي أسطر الكود كما يمكننا الاستفادة من العديد من الخصائص الأخرى للفئة Process مثل ExitCode التي تعيد القيمة المعادة من العملية عند انتهائها

إذا أردنا فتح موقع على الويب باستخدام المتصفح الافتراضي

باستخدام Process.start يمكننا عمل ذلك مباشرة

```
Process.Start("www.aya.sy")
```

وبطريقة ProcessStartInfo

```
Process.Start(New ProcessStartInfo("www.aya.sy"))
```

والسبب في ذلك أن الوظيفة Process.Start ذكية بحيث يمكنها تشغيل أي ملف - وثيقة وورد مثلا أو عنوان موقع انترنت باستخدام البرنامج الافتراضي المرتبط به مباشرة فلكي نقوم بفتح ملف docx مثلا باستخدام البرنامج الافتراضي المرتبط به وهو في هذه الحالة Word 2007 يكفي أن نمرر اسم الملف مع المسار الكامل له

```
Process.Start("C:\Users\SamerSelo\Documents\أطباق متنوعة.docx")
```

إذا أردنا إرسال بريد الكتروني لبريد ما مثلا info@aya.sy يمكننا عمل ذلك أيضا باستخدام Process.Start الذي يفتح لنا محرر البريد الالكتروني الافتراضي عند تمرير عنوان البريد الالكتروني لتلك الوظيفة مثال

```
Process.Start("mailto:info@aya.sy")
```

الأصوات في VB .net

قبل فيجول ستوديو 2005 كانت عملية إضافة أبسط الأصوات إلى برنامجك تعتبر تحديا نوعا ما ولكن تمت إضافة مجموعة هائلة من الفئات ومجالات الأسماء في الفريموورك 2 وبعضها سيساعدك في عمل ذلك وسوف أقوم بنظرة على المجال System.Media في هذا الموضوع

لإطلاق صوت الصافرة beep سنحتاج لاستخدام الفئة SystemSounds والتي لها خمس خصائص عامة Asterisk, Beep, Exclamation, Hand, Question كل من هذه الخصائص يقوم بعرض غرض object يعود إلى SystemSounds ويعرض إجرائية Playلجعل الكمبيوتر يصدر ذلك الصوت ولتشغيل صوت beep الخاص بالنظام نستخدم

```
System.Media.SystemSounds.Beep.Play
```

والشئ الجميل في SystemSounds هو أنه عندما يقوم مستخدم النظام بتغيير أصوات النظام فهذا سينعكس على برنامجك فورا فعلى سبيل المثال عندما تريد عرض صندوق الرسائل MessageBox يمكنك تشغيل صوت النظام المناسب ليصاحبه وذلك حتى تلفت انتباه مستخدم تطبيقك لمحتويات الرسالة

```
System.Media.SystemSounds.Exclamation.Play()
MessageBox.Show("Visual Basic!", "Advanced Basics", _
                 MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
```

تشغيل الأصوات يبدو مفيدا ولكن هناك المزيد فعندما تريد عرض أصوات معقدة أكثر ففي System.Media ستجد الفئة SoundPlayer التي تقوم بتشغيل ملف WAV وذلك من أماكن مختلفة كملف على القرص الصلب أو موقع إنترنت أو حتى مصادر مضمنة ضمن exe البرنامج. وأول خطوة تقوم بها هي تحميل الملف فإذا كنت تقوم بتشغيله من ملف أو من موقع انترنت يجب عليك القيام بتحميله أولا أما عندما تريد تشغيله من مصادر مضمنة فإجرائية التشغيل ستقوم بتحميله بالنيابة عنك إن لم تقم بتحميله أولا وبهذا يكون لديك مجال كبير من المرونة للتشغيل المتزامن أو غير المتزامن
.
فعندما تريد تشغيل ملف wav بوقت قريب لوقت التشغيل ولا تستطيع ضمان اكتمال التحميل بوقت تشغيل الإجراء play عندها يجب عليك اختيار التشغيل المتزامن وسيئته هو أنك سوف تمنع استكمال تنفيذ التطبيق في ذلك المسار thread حتى انتهاء التحميل والتشغيل غير المتزامن فعال خاصة عندما تقوم بالتحميل من مصدر بطئ مثل عنوان انترنت مثلا والملف كبير نوعا ما عندها يجب عليك استخدام التحميل الغير متزامن الذي سيسمح لبرنامجك باستكمال التنفيذ بنفس الوقت الذي يقوم فيه بتحميل الملف وسوف يتم إعلامك بانتهاء التحميل بإطلاق الحدث LoadCompleted وذلك في حالة رغبتك بإجراء معالجة إضافية أو إجراء أي عمل آخر بالإضافة إلى ذلك يمكنك استخدام الخاصية IsLoadCompletedفي أي وقت للتحقق من حالة التحميل ويمكن أن يبدو كودك في حالة التحميل الغير متزامن كالكود التالي

```
Dim Player As New SoundPlayer
Player.SoundLocation = "C:\Program Files\Messenger\newemail.Wav"
Player.LoadAsync()
```

الآن أنت جاهز لتشغيل ملف wav وهنا أيضا لديك الخيار بالتشغيل المتزامن أو الغير متزامن تماما كالتحميل. فالتشغيل المتزامن يجبر مستخدم تطبيقك على الاستماع للملف كاملا قبل أن يكمل التطبيق مسار تنفيذه ولن تضطر للقلق عبر أي تدخل لكود أو مستخدم في ذلك الوقت فعندما يكون طول ملفك ثانية أو اثنتين لن يشكل ذلك مشكلة ولكن أي زمن أطول من ذلك سيسبب اختناقا للتطبيق وسوف يشكك المستخدم في جدوى التطبيق وفي الواقع الإجراء SoundPlayer.Play سيقوم بتشغيل الملف بشكل غير متزامن حتى لا يعيق تنفيذ برنامجك

```
Dim Player As New SoundPlayer
Player.SoundLocation = "C:\Program Files\Messenger\newemail.Wav"
Player.LoadAsync()
If Player.IsLoadCompleted Then Player.Play()
```

وإذا رغبت في التشغيل المتزامن ستضطر لاستدعاء الاجراء SoundPlayer.PlaySync ولكن هذا سيبطئ تنفيذ برنامج بشكل كبير أثناء تشغيل الملف وإذا رغبت في أن يستمر تشغيل الملف حتى يقوم المستخدم بضغط زر ما أو حتى تنتهي من عملية برمجية معينة عندها ستستدعي الإجرائية PlayLooping فستقوم الفريموورك بتشغيل الملف بشكل مستمر بشكل غير متزامن حتى تقوم باستدعاء الإجرائية

SoundPlayer.Stopوإذا قمنا بإلقاء نظرة على المجال My في الفريموورك سنجد أن جميع ما تمت مناقشته حتى الآن موجود ضمن My.Computer.Audioفلتشغيل ملف يمكنك استخدام

```
My.Computer.Audio.Play("C:\Program Files\Messenger\newemail.Wav")
```

ولا حاجة للقلق حول إنشاء غرض object لـ SoundPlayer أو التحميل من مسار إنترنت أو ملف ... الخ إذا لماذا قمنا بكتابة كود أكثر بهذا الشكل؟ بالإضافة إلى تعلم كيفية إجراء ذلك ضمن الفريموورك سوف تحتاج للعمل مع الفئة SoundPlayerإذا كنت ستحتاج للتعامل مع أي من الأحداث الثلاثة التي تطلقها تلك الفئة
أولا الحدث LoadCompleted الذي ذكرناه سابقا سوف يطلق في نهاية التحميل الغير متزامن للملف فإذا كنت تحمل من عنوان انترنت أو ملف ستجد انه هذا الحدث هام جدا لالتقاطه وسوف تتحقق من محددات الحدث للتحقق من اكتمال التحميل بنجاح

```
Private Sub Player_LoadCompleted(ByVal sender As Object, _
        ByVal e As ComponentModel.AsyncCompletedEventArgs) _
        Handles Player.LoadCompleted
    If e.Error IsNot Nothing Then
        ' Handle error
        isLoaded = False
    Else
        isLoaded = True
    End If
End Sub
```

و الحدث SoundLocationChanged سوف يتم إطلاقه عندما يتم ربط مصدر صوتي جديد لـ SoundPlayer عندها كي تقوم بما ترغب كتحميل الملف الجديد بشكل غير متزامن أو إيقاف تشغيل الملف الحالي أو القيام بأي عمل آخر يحتاجه تطبيقك والحدث الأخير الهام هو StreamChangedوهو مشابه للحدث السابق ولكنه مفيد بشكل خاص عندما تقوم بتحميل الملف من الذاكرة أو ملف على القرص MemoryStream or FileStreamوإذا رغبت في القيام بأكثر من تشغيل ملف wav عندها ستضطر للخروج خارج الفريموورك وأحد الخيارات هو اللجوء إلى Windows Media Player وبتحميل Windows Media Player SDK مجانا من موقع مايكروسوفت يمكنك عندها تشغيل تقريبا أي صيغة ملف وببنية بسيطة مشابهة لـلمجال System.Media ستجد أن التعامل مع Windows Media Player سهلا للغاية فمثلا يمكنك استخدام كود شبيه بالتالي لتشغيل أي ملف يمكن تشغيله بواسطة Windows Media Player وذلك بعد تنصيب المكتبة المذكورة وإضافة المرجع المناسب لها في برنامجك

```
Dim WMP As New WMPLib.WindowsMediaPlayer
WMP.URL = "C:\My Music\Funky Cold Medina.wma"
WMP.controls.play()
```

أصبح لديك الآن ثلاثة طرق لإضافة الأصوات إلى برنامجك الأولى باستخدام مجال الأسماء الجديد System.Media من أو استخدام المجال MY الجديد الخاص بفيجول بايزيك من أجل تشغيل ملفات WAV أو استخدام Windows Media Player لدعم جميع أنواع صيغ الملفات

إدخال وإخراج الأقراص القابلة للنزع برمجيا Eject/Load Removable Media

يمكن ذلك باستخدام الكود التالي

```
Public Module EjectLoadRemovable

   Private Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" _
     (ByVal lpFileName As String, ByVal dwDesiredAccess As Integer, _
     ByVal dwShareMode As Integer, ByRef lpSecurityAttributes As Long, _
     ByVal dwCreationDisposition As Integer, ByVal dwFlagsAndAttributes _
     As Integer, ByVal hTemplateFile As Integer) As Integer

   Private Declare Function DeviceIoControl Lib "kernel32" (ByVal hDevice As _
     Integer, ByVal dwIoControlCode As Integer, ByRef lpInBuffer As Object, _
     ByVal nInBufferSize As Integer, ByRef lpOutBuffer As Object, ByVal _
     nOutBufferSize As Integer, ByRef lpBytesReturned As Integer, ByRef _
     lpOverlapped As Object) As Integer

   Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject _
    As Integer) As Integer

   Private Const INVALID_HANDLE_VALUE As Short = -1
   Private Const OPEN_EXISTING As Short = 3
   Private Const FILE_FLAG_DELETE_ON_CLOSE As Integer = 67108864
   Private Const GENERIC_READ As Integer = &H80000000
   Private Const GENERIC_WRITE As Integer = &H40000000
   Private Const IOCTL_STORAGE_EJECT_MEDIA As Integer = 2967560
   Private Const IOCTL_STORAGE_LOAD_MEDIA As Integer = 2967564
   Private Const IOCTL_STORAGE_LOAD_MEDIA2 As Integer = 2951180
   Private Const VWIN32_DIOC_DOS_IOCTL As Short = 1

   ' To Eject Removable media just pass the drive letter to the sub
   Public Sub EjectRemovable(ByVal EjectDrive As String)
       Dim hDrive, DummyReturnedBytes As Integer
       Dim DriveLetterAndColon As String

       'Make it all caps for easy interpretation
       DriveLetterAndColon = UCase(Left(EjectDrive & ":", 2))
       hDrive = CreateFile("\\.\" & DriveLetterAndColon, GENERIC_READ Or _
                           GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0)

       If hDrive <> INVALID_HANDLE_VALUE Then
           'Eject media!
           Call DeviceIoControl(hDrive, IOCTL_STORAGE_EJECT_MEDIA, 0, 0, 0, _
                                0, DummyReturnedBytes, 0)

           Call CloseHandle(hDrive) 'Clean up after ourselves
       End If

   End Sub

   ' To Load Removable media just pass the drive letter to the sub
   Public Sub LoadRemovable(ByVal EjectDrive As String)
       Dim hDrive, DummyReturnedBytes As Integer
```

```
        Dim DriveLetterAndColon As String

        'Make it all caps for easy interpretation
        DriveLetterAndColon = UCase(Left(EjectDrive & ":", 2))
        hDrive = CreateFile("\\.\" & DriveLetterAndColon, GENERIC_READ Or _
                            GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0)

        If hDrive <> INVALID_HANDLE_VALUE Then
            'Eject media!
            Call DeviceIoControl(hDrive, IOCTL_STORAGE_LOAD_MEDIA, 0, 0, 0, _
                                  0, DummyReturnedBytes, 0)

            Call CloseHandle(hDrive) 'Clean up after ourselves
        End If
    End Sub

End Module
```

وهذا مثال عن الاستخدام

```
EjectRemovable("h:\")

LoadRemovable("h:\")
```

وهذه طريقة أخرى باستخدام mciSendString

```
Module EjectLoadUsingMCI

    <DllImport("winmm", EntryPoint:="mciSendStringA")> _
           Private Sub mciSendString(ByVal lpszCommand As String, _
           ByVal lpszReturnString As String, ByVal cchReturnLength As Integer, _
           ByVal hwndCallback As Integer)

    End Sub

    Public Sub OpenCDDoor(ByVal Drive As String)
        mciSendString("open cdaudio!" & Drive, "", 0, 0)
        mciSendString("Set " & Drive & " door open wait", "", 0, 0)
    End Sub

    Public Sub CloseCDDoor(ByVal drive As String)
        mciSendString("open cdaudio!" & drive, "", 0, 0)
        mciSendString("Set " & drive & " door closed wait", "", 0, 0)
    End Sub

End Module
```

ولكن تتميز الطريقة الأولى عن الثانية بأنها صالحة لجميع أنواع الأقراص القابلة للنزع Removable بينما الطريقة الثانية صالحة للأقراص من النوع CDRom فقط كما أنه عند استخدام الطريقة الثانية لا يعود بإمكانك برمجيا فتح وإغلاق باب السواقة إلا باستخدام نفس الطريقة حتى تعيد إقلاع الجهاز

مراقبة نظام الملفات - التحكم FileSystemWatcher

توفر لك الأداة FileSystemWatcher إمكانية مراقبة مجلد أو شجرة مجلدات بحيث تحصل على تنبيهات عندما يحصل أي شئ بداخلها فمثلا عندما يتم إنشاء أو حذف أو إعادة تسمية مجلد أو ملف أو عندما تتغير خصائص مجلد ما حيث تعتبر هذه الأداة ذات فائدة في العديد من الظروف فإذا كنت تقوم بعمل برنامج يقوم بتشفير الملفات آليا عندما يتم تخزينها في مجلد معين فبدون هذه الأداة سيتوجب عليك استطلاع ذلك المجلد على فترات زمنية محددة – عادة باستخدام الـ Timer – ولكن هذه الأداة تسهل عليك هذا الأمر. وممكن أن تكون هذه الأداة مفيدة عندما تقوم بتخزين ملف بيانات في الذاكرة لتسهيل الوصول إليه بسرعة ولكنك تحتاج لإعادة تحميله في الذاكرة إذا قام برنامج آخر بتغيير محتوياته.

تعريف التحكم FileSystemWatcher

يمكنك إنشاء التحكم FileSystemWatcher بإحدى طريقتين إما عن طريق الكود أو بسحبها من شريط الأدوات إلى النموذج من خلال مصمم النماذج ولا يوجد أي اختلاف في الأداء بينهما فيمكنك استخدام أي طريقة تفضلها وإنشاؤه عن طريق الكود يتم بسهولة

```
' Use WithEvents to be able to trap events from this object.
Dim WithEvents fsw As New FileSystemWatcher()
```

قبل استخدامك لهذا العنصر عليك أولا تعريف الخصائص Path و IncludeSubdirectories و Filter و NotifyFilter

فالخاصية Path تحدد مسار المجلد الذي ترغب بمراقبته لاحظ أنه سيتم إعلامك بالتغييرات داخل المجلد وليس بالتغييرات على خصائصه

والخاصية IncludeSubdirectories يجب ضبطها إلى False إذا كنت ترغب بإعلامك بالتغييرات داخل ذلك المجلد فقط أو إلى True إذا كنت ترغب بمراقبة كامل لشجرة المجلد بحيث يكون جذرها هو المجلد المحدد بالخاصية Path

والخاصية Filter تمكنك من تحديد ما هي الملفات التي تهمك فاستخدام *.* يجعلك تراقب جميع الملفات بينما استخدام *.txt يجعلك تراقب الملفات التي تملك اللاحقة txt فقط والقيمة الافتراضية لهذه الخاصية هي Null وهي تطابق *.*

والخاصية NotifyFilter مرمزة على مستوى البت bit-coded بحيث يمكنك تحديد تشكيلة من واحد أو أكثر من قيم مرقمة enumerated كـ Attributes و CreationTime و DirectoryName و FileName و LastAccess و LastWrite و Security و Size و القيمة الافتراضية لها هي LastWrite Or FileName Or DirectoryName ففي هذه الحالة لن تتلقى تنبيهات عندما تتغير الخصائص

وإليك مثال عن كيفية تعريف تحكم FileSystemWatcher من أجل التغييرات في شجرة المجلد C:\Windows

```
Dim WithEvents fsw As New FileSystemWatcher()
…
fsw.Path = "c:\windows"
fsw.IncludeSubdirectories = True            ' Watch subdirectories.
fsw.Filter = "*.dll"                        ' Watch only DLL files.
' Add attribute changes to the list of changes that can fire events.
fsw.NotifyFilter = fsw.NotifyFilter Or NotifyFilters.Attributes
' Enable event notification.
fsw.EnableRaisingEvents = True
```

الحصول على التنبيهات

بعد أن تكون قد عرفت هذا العنصر بصورة صحيحة ستحصل على التنبيها   ت فور حدوث أي شئ حيث يمكنك تفعيل ذلك بمعالجة الأحداث أو باستخدام الطريقة WaitForChange

وأبسط طريقة للحصول على التنبيهات من FileSystemWatcher هي بكتابة معالجات للأحداث الخاصة بهذا العنصر ولكن هذه الأحداث لن تنطلق حتى تقوم بضبط الخاصية EnableRaisingEvents إلى True و تستقبل الأحداث Created و Deleted و Changed المحدد FileSystemEventArgs الذي يعرض خاصيتان هامتان Name اسم الملف الذي وقع عليه الحدث و FullPath مسار ذلك الملف الكامل

```
Private Sub fsw_Created(ByVal sender As Object, _
      ByVal e As FileSystemEventArgs) Handles fsw.Created
    Console.WriteLine("File created: {0}", e.FullPath)
End Sub

Private Sub fsw_Deleted(ByVal sender As Object, _
       ByVal e As FileSystemEventArgs) Handles fsw.Deleted
    Console.WriteLine("File deleted: {0}", e.FullPath)
End Sub

Private Sub fsw_Changed(ByVal sender As Object, _
       ByVal e As FileSystemEventArgs) Handles fsw.Changed
    Console.WriteLine("File changed: {0}", e.FullPath)
End Sub
```

المحدد FileSystemEventArgs يعرض أيضا خاصية ChangeType وهي خاصية مرقمة Enumerated حيث تخبرك بالحدث الذي وقع على الملف بحيث يمكنك استخدامها لعمل إجراء معالجة وحيد للأحداث الثلاثة السابقة

```
Private Sub fsw_All(ByVal sender As Object, ByVal e As FileSystemEventArgs) _
      Handles fsw.Changed, fsw.Created, fsw.Deleted
    Console.WriteLine("File changed: {0} ({1})", e.FullPath, e.ChangeType)
End Sub
```

الحدث Changed لا يستقبل أية معلومات حول نوع التغيير الذي أطلق الحدث والحدث Renamed يستقبل المحدد RenamedEventArgs الذي يعرض خاصيتان إضافيتان هما OldName الاسم القديم و OldFullPath المسار الكامل القديم

```
Private Sub fsw_Renamed(ByVal sender As Object, ByVal e As RenamedEventArgs) _
     Handles fsw.Renamed
    Console.WriteLine("File renamed: {0} => {1}", e.OldFullPath, e.FullPath)
End Sub
```

ويمكن أن يكون لديك أكثر من FileSystemWatcher واحد يمكنك معالجة أحداثهم جميعا بنفس إجراء معالجة حدث واحد وعند ذلك يمكنك استخدام المحدد الأول sender لمعرفة أي FileSystemWatcher أطلق الحدث.

يطلق FileSystemWatcher حدث مستقل لكل حدث يقع على ملف ما فعلى سبيل المثال إذا حذفت 10 ملفات ستتلقى 10 أحداث بحيث يكون لكل ملف الحدث الخاص به وكذلك إن قمت بنقل 10 ملفات من مجلد لآخر ستتلقى 10 أحداث حذف و 10 أحداث إنشاء

الطريقة WaitForChanged

إذا كان برنامجك لا يقوم بعمل أي شئ سوى انتظار التغييرات في مجلد معين أو أنك تقوم بمراقبة تغييرات الملفات من مسار ثانوي Secondary Thread يمكنك كتابة كود أبسط وأكثر فاعلية باستخدام الطريقة WaitForChanged وهي متزامنة Synchronous أي أنها لا تعود حتى يحدث تغيير ما أو ينتهي الوقت المحدد timeout وعندما تعود هذه الطريقة يستقبل البرنامج النتيجة عبر التركيب WaitForChangedResult Structure الذي تمكنك خصائصه من تحديد فيما إذا انتهت المدة أو أن حدث ما قد وقع واسم الملف المتعلق بالحدث

```
' Create a *new* FileSystemWatcher component with values from
' the txtPath and txtFilter controls.
Dim tmpFsw As New FileSystemWatcher(txtPath.Text, txtFilter.Text)
' Wait max 10 seconds for any file event.
Dim res As WaitForChangedResult = tmpFsw.WaitForChanged(WatcherChangeTypes.All, 10000)

' Check whether the operation timed out.
If res.TimedOut Then
   Console.WriteLine("10 seconds have elapsed without an event")
Else
    Console.WriteLine("Event: {0} ({1}), res.Name, res.ChangeType.ToString())
End If
```

الطريقة WaitForChanged تراقب فقط التغييرات في المجلد المشار إليه وتتجاهل الخاصية IncludeSubDirectories ولهذا السبب فالتركيب WaitForChangedResult Structure يعيد حقل Name فقط ولا يعيد الحقل FullPath والمحدد الأول الذي تقوم بتمريره إلى الطريقة WaitForChanged يمكنك من تحديد نوع العملية التي تريد اعتراضها

```
' Pause the application until the c:\temp\temp.dat file is deleted.
tmpFsw = New FileSystemWatcher("c:\temp", "temp.dat")
tmpFsw.WaitForChanged(WatcherChangeTypes.Deleted)
```

Buffer Overflows

يجب عليك إدراك المشاكل المحتملة نتيجة انطلاق العديد من الأحداث خلال فترة قصيرة فالعنصر FileSystemWatcher يستخدم buffer داخلي ليتتبع أفعال نظام الملفات حتى لو كان البرنامج لا يستطيع تخديم تلك الأحداث بالسرعة الكافية وبالوضع الافتراضي يكون حجم الـ buffer هو 8KB ويمكنه تخزين حتى 160 حدث فكل حدث يأخذ 16 بايت إضافة لـ 2 بايت لكل محرف في اسم الملف وإن كنت تتوقع نشاطا أكثر فعليك زيادة حجم الـ buffer بضبط الخاصية InternalBufferSize إلى قيمة أكبر والحجم يجب أن يكون عددا صحيحا Integer مضروبا بحجم صفحة النظام (4KB في حالة ويندوز 2000 وما تلاه) كما يمكنك استخدام الخاصية NotifyFilter لتحديد عدد عمليات التغيير التي تطلق الحدث Changed أو ضبط الخاصية IncludeSubdirectories إلى False إن كنت لا تحتاج لمراقبة شجرة المجلد كاملة (يمكنك استخدام عدة عناصر FileSystemWatcher لمراقبة عدد من المجلدات إن كنت لا تحتاج لمراقبة الشجرة كاملة تحت مسار مجلد معين) ولا يمكنك استخدام الخاصية Filter لمنع Buffer Overflowing لأنها تقوم بتصفيتها بعد أن يتم إضافتها لـ Buffer وعندما يحدث فيضان في الـ buffer سينطلق حدث خطأ

```
Private Sub fsw_Error(ByVal sender As Object, ByVal e As ErrorEventArgs) _
      Handles fsw.Error
   Console.WriteLine("FileSystemWatcher error: {0}", e.GetException().Message)
End Sub
```

فإن لاحظت أن برنامجك يتلقى هذا الحدث فيجب عليك تغيير إستراتيجية معالجة الأحداث لديك فمثلا يمكنك تخزين الأحدث في صف queue ثم تخديمهم عبر مسار thread آخر

معالجة المشاكل

في الحالة الافتراضية فأحداث FileSystemWatcher يتم تنفيذها على مسار مأخوذ من بركة مسارات النظام System thread pool وبما أن تحكمات Windows Forms ليست أمنة للمسارات not thread safe لذلك يجب أن لا تحاول الوصول إلى أي تحكم أو حتى النموذج نفسه من داخل أحداث التحكم FileSystemWatcher وإن وجدت هذا التصرف غير مقبول فعليك تمرير تحكم Windows Forms للخاصية SynchronizingObject كالمثال

```
' Use the Form object as the synchronizing object.
fsw.SynchronizingObject = Me
```

فالكود السابق يتأكد من أن جميع الأحداث يتم تنفيذها بنفس المسار الذي يخدم النموذج نفسه وفي Visual Studio 2005 عندما تستخدم مصمم النماذج لإنشاء العنصر FileSystemWatcher يتم ضبط هذه الخاصية تلقائيا للنموذج الذي يضم التحكم

وفيما يلي بعض الأفكار المفيدة عند التعامل مع التحكم FileSystemWatcher وبعض المشاكل التي قد تحتاج لحلها عندما تقوم باستخدامه

- يقوم التحكم بالبدء بإطلاق الأحداث عندما تكون الخاصية Path غير فارغة والخاصية EnableRaisingEvents مضبوطة إلى True كما يمكنك منعه من إطلاق الأحداث خلال طور التحميل للنموذج بتضمين أوامر التهيئة الخاصة بك بين العبارتين BeginInit و EndInit كما يفعل مصمم النماذج الخاص بفيجوال ستوديو

- في بعض الحالات قد تستقبل عدة أحدث إنشاء للملف اعتمادا على الطريقة التي يتم إنشاء الملف بها فمثلا عندما تقوم بإنشاء ملف باستخدام المفكرة Notepad ستلاحظ سلسلة من الأحداث Created و Deleted و Created و Changed

- التغيير في ملف يولد أيضا حدث إضافي في المجلد الأب أيضا لأن المجلد يحافظ على معلومات حول الملفات الموجودة داخله

- إذا كان المجلد المحدد كقيمة للخاصية Path تمت إعادة تسميته فإن التحكم FileSystemWatcher يستمر بالعمل بصورة صحيحة ولكن في هذه الحالة يستمر التحكم بإعادة الاسم القديم للمجلد وبالتالي قد تحصل على خطأ عند استخدامه وهذا يحدث بسبب أن التحكم يتعامل مع مقبض المجلد الذي لا يتغير بتغيير اسم المجلد

- إذا أنشأت مجلد ضمن المجلد الذي تتم مراقبته وكانت الخاصية IncludeSubdirectories مضبوطة على True فستتم مراقبة المجلد الجديد أيضا

- عندما يتم إنشاء ملف كبير ضمن المجلد فقد لا تستطيع قراءة كامل الملف مباشرة لأنه يكون ما يزال مملوكا من قبل العملية Process التي قامت بكتابة البيانات إلى ذلك الملف ويجب عليك حماية الكتابة إلى الملف الأصلي باستخدام حلقة Try فإن تم إطلاق خطأ يمكنك محاولة العملية بعد بضع ميلي ثانية أخرى

- عندما يقوم المستخدم بحذف ملف في مجلد فإن ملفا جديدا سيتم إنشاؤه في مجلد سلة المحذوفات RecycleBin

القسم الخامس – الانترنت

ويضم المواضيع التالية:

- إضافة وصلات ويب وبريد الكتروني لنافذتك
- الاتصال بالانترنت برمجيا
- كيف نستخدم My.Computer.Network لرفع وتحميل ملفات في Visual Basic 2005
- منع تغيير الصفحة الافتراضية للإنترنت إكسبلورر و تغييرها برمجيا

الأداة LinkLabel

إضافة وصلات ويب وبريد الكتروني لنافذتك

تمكنك هذه الأداة من إضافة وصلات ويب لمشروعك كما يمكنك تحديد وصلة أو أكثر ضمن النص الظاهر فيه

- ضع LinkLabel على النافذة حتى نقوم بالتجريب

١. اضبط الخاصية Text لـ LinkLabel إلى أي عنوان موقع مثلا www.arabteam2000.com ثم انتقل إلى محرر الكود وفي الخاصية LinkClicked للتحكم LinkLabel اكتب الكود التالي حيث استخدمنا Process.Start بالصيغة Process.Start (String) التي تقوم بتشغيل برنامج أو وثيقة حسب السلسلة النصية الممررة لها لفتح الموقع المراد

```
Process.Start(Me.LinkLabel1.Text)
```

٢. ولتحديد وصلة لإرسال بريد الكتروني اضبط الخاصية Text إلى أي عنوان بريد الكتروني تريد مثلا someone@yahoo.com وعدل الكود السابق إلى

```
Process.Start("mailto:" & Me.LinkLabel1.Text)
```

٣. كما يمكن إضافة العديد من الوصلات في نفس التحكم وذلك بجعل القيمتان Start و Length المرتبطتين بالخاصية LinkArea إلى الصفر وإضافة الوصلات إلى الخاصية Links برمجيا حيث نستخدم الطريقة Add للخاصية Links للتحكم LinkLabel لإضافة الوصلات للتحكم حيث تملك الطريقة Add ثلاث طرائق Overloaded وهي

- إضافة وصلة من نوع LinkLabel.Link

LinkLabel.Links.Add (LinkLabel.Link)

- إضافة وصلة بتحديد مجال محارف من النص الظاهر في التحكم

LinkLabel.Links.Add (Int32, Int32)

- إضافة وصلة بتحديد مجال محارف من النص الظاهر في التحكم كاسم للوصلة وتمرير عنوان الوصلة كمتغير Object

LinkLabel.Links.Add (Int32, Int32, Object)

٤. اضبط الخاصية text للتحكم إلى Try at Yahoo or at Arab team or Email me و ضع القيم Start و Length المرتبطتين بالخاصية LinkArea كلاهما إلى الصفر ثم في الحدث Load للنموذج ضع الكود التالي حيث استخدمنا الصيغة الأخيرة للطريقة Add

```
With Me.LinkLabel1
     .Links.Add(7, 5, "www.yahoo.com")
     .Links.Add(19, 9, "www.arabteam2000.com")
     .Links.Add(32, 8, "mailto:someone@yahoo.com")
End With
```

٥. الآن لتشغيل كل وصلة عند النقر عليها غير الكود الموجود ضمن الحدث LinkClicked للتحكم LinkLabel إلى

```
Dim Url As String = CStr(e.Link.LinkData)
Process.Start(Url)
```

وهنا لم نعد نستخدم الخاصية Text للتحكم كما فعلنا في المثال الأول بل نستخدم خصائص المحدد e الذي هو من النوع LinkLabelLinkClickedEventArgs للحصول على عنوان الموقع أو البريد الالكتروني الذي نريده وذلك من خلال الخاصية e.Link.LinkData التي هي من النوع Object حيث نقوم بتحويلها إلى String قبل تمريرها لوظيفة Process.start كقيمة نصية

الاتصال بالانترنت برمجيا

تأسيس الاتصال بالانترنت

لتأسيس الاتصال بالانترنت نستخدم الدالة InternetAutodial وذلك لتأسيس الاتصال باستخدام الاتصال الافتراضي للويندوز ويكون تعريفها كالتالي:

```
Private Declare Auto Function InternetAutodial Lib "wininet.dll" _
    (ByVal dwFlags As Long, ByVal hwndParent As Integer) As Boolean
```

حيث يمكن للمحدد dwFlags أن يحمل إحدى القيم التالية:

```
Private Const INTERNET_AUTODIAL_FORCE_ONLINE = 1
Private Const INTERNET_AUTODIAL_FORCE_UNATTENDED = 2
Private Const INTERNET_AUTODIAL_FAILIFSECURITYCHECK = 4
Private Const INTERNET_AUTODIAL_OVERRIDE_NET_PRESENT = 8
```

وهذا مثال عن استخدام الدالة:

```
If InternetAutodial(0, Me.Handle.ToInt32) Then
    MsgBox("Internet is connected Now", MsgBoxStyle.Exclamation)
Else
    MsgBox("Internet is not connected Now", MsgBoxStyle.Critical)
End If
```

قطع الاتصال بالانترنت

لقطع الاتصال بالانترنت نستخدم إحدى الدالتين (InternetAutodialHangup و InternetHangup) حيث يكون تعريفهما كالتالي:

```
Private Declare Auto Function InternetAutodialHangup Lib "wininet.dll" _
    (ByVal dwReserved As Long) As Boolean

Private Declare Auto Function InternetHangUp Lib "wininet.dll" _
    (ByVal dwConnection As Long, ByVal dwReserved As Long) As Boolean
```

حيث يكون المحدد dwReserved في كلتا الدالتين مساويا للصفر والبارمتر dwConnection في الدالة الثانية يحمل رقم الاتصال الذي نريد قطعه

معرفة خصائص الاتصال

يمكن معرفة العديد من خصائص الاتصال الحالي بالانترنت عن طريق الدالة InternetGetConnectedStateEx وتعريفها يكون كالتالي:

```
Private Declare Function InternetGetConnectedStateEx Lib "wininet.dll" _
    (ByRef lpdwFlags As Long, ByVal lpszConnectionName As String, _
    ByVal dwNameLen As Integer, ByVal dwReserved As Long) As Boolean
```

والمحدد lpdwFlags يحمل قيمة معادة بواحدة أو أكثر من القيم التالية:

```
Private Const INTERNET_CONNECTION_MODEM = &H1
Private Const INTERNET_CONNECTION_LAN = &H2
Private Const INTERNET_CONNECTION_PROXY = &H4
Private Const INTERNET_CONNECTION_MODEM_BUSY = &H8
Private Const INTERNET_RAS_INSTALLED = &H10
Private Const INTERNET_CONNECTION_OFFLINE = &H20
Private Const INTERNET_CONNECTION_CONFIGURED = &H40
```

و بافتراض أنه لديك نافذة عليها زر أوامر و ثلاثة صناديق نصوص وصندوق قائمة إليك سرد لحدث الضغط على الزر الذي سيقوم بإظهار معلومات الاتصال:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click

    ListBox1.Items.Clear()
    Try
        Dim sConnType As String = Space(256)
        Dim Ret As Long

        TextBox1.Text = InternetGetConnectedStateEx(Ret, sConnType, _
                sConnType.Length, 0)

        If Ret And INTERNET_CONNECTION_MODEM Then
            ListBox1.Items.Add("MODEM")
        End If
        If Ret And INTERNET_CONNECTION_LAN Then
            ListBox1.Items.Add("LAN")
        End If
        If Ret And INTERNET_CONNECTION_CONFIGURED Then
            ListBox1.Items.Add("CONFIGURED")
        End If
        If Ret And INTERNET_CONNECTION_MODEM_BUSY Then
            ListBox1.Items.Add("MODEM BUSY")
        End If
        If Ret And INTERNET_CONNECTION_OFFLINE Then
            ListBox1.Items.Add("OFFLINE")
        End If
        If Ret And INTERNET_CONNECTION_PROXY Then
            ListBox1.Items.Add("PROXY")
        End If
        If Ret And INTERNET_RAS_INSTALLED Then
            ListBox1.Items.Add("RAS INSTALLED")
        End If

        TextBox2.Text = Hex(Ret)
        TextBox3.Text = sConnType
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

طريقة أخرى لمعرفة هل يوجد اتصال أم لا

يمكنك استخدام الاجرائية التالية:

```
Public Function IsConnectionAvailable() As Boolean
    ' Returns True if connection is available
    ' Replace www.yoursite.com with a site that
    ' is guaranteed to be online - perhaps your
    ' corporate site, or microsoft.com
    Dim objUrl As New System.Uri("http://www.yoursite.com/")
    ' Setup WebRequest
    Dim objWebReq As System.Net.WebRequest
    objWebReq = System.Net.WebRequest.Create(objUrl)
    Dim objResp As System.Net.WebResponse
    Try
        ' Attempt to get response and return True
        objResp = objWebReq.GetResponse
        objResp.Close()
        objWebReq = Nothing
```

```
        Return True
    Catch ex As Exception
        ' Error, exit and return False
        objResp.Close()
        objWebReq = Nothing
        Return False
    End Try
End Function
```

كيف نستخدم My.Computer.Network لرفع وتحميل ملفات في Visual Basic 2005

موجز

تعلم كيفية استخدام My.Computer.Network لرفع وتحميل الملفات ضمن شبكة في فيجول بايزيك 2005. وهذا المثال يحتوي على مثال كود بالخطوات يريك كيف يقوم My.Computer.Network بهذه المهمات

تقديم

هذا المقال يصف كيفية استخدام My.Computer.Network لرفع وتحميل الملفات عبر شبكة باستخدام فيجول بايزيك 2005وباستخدامه يمكنك نقل الملفات من مصادر شبكة بعيدة للكمبيوتر المحلي ولعمل ذلك ستستخدم الطرائق التالية في تطبيق فيجول بايزيك:

- My.Computer.Network.DownloadFile تحمل ملف محدد بعيد وتحفظه في المكان المحدد في الموقع المحدد على الكمبيوتر الحالي
- My.Computer.Network.UploadFile ترسل ملف محدد من الكمبيوتر المحلي إلى عنوان مضيف محدد

استخدام الطريقة My.Computer.Network.DownloadFile لتحميل الملفات

```
Try
    '' Without Progress
    ' My.Computer.Network.UploadFile("<LocalPath>", "<HostPath>")
    My.Computer.Network.UploadFile("C:\Upload\test.txt", " \\Server\test.txt")

    '' With Progress
    ' My.Computer.Network.UploadFile("<LocalPath>", "<HostPath>", _
    '     "", "", True, 500)
    My.Computer.Network.UploadFile("C:\Upload\test.txt", " \\Server\test.txt", _
       "", "", True, 500)

    MessageBox.Show("File uploaded.")
Catch ex As Exception
    MessageBox.Show("Access failed" & vbCrLf & ex.Message)
End Try
```

استخدام My.Computer.Network.UploadFile لرفع الملفات

```
Try
    '' Without Progress
    ' My.Computer.Network.DownloadFile("<HostPath>", "<LocalPath>")
    My.Computer.Network.DownloadFile("\\Server\test.txt", "
C:\Download\test.txt")

    '' With Progress
    ' My.Computer.Network.DownloadFile("<HostPath>", "<LocalPath>", _
    '     "", "", True, 500, True)
    My.Computer.Network.DownloadFile("\\Server\test.txt", "
C:\Download\test.txt", _
       "", "", True, 500, True)

    MessageBox.Show("File downloaded.")
Catch ex As Exception
```

```
    MessageBox.Show("Access failed" & vbCrLf & ex.Message)
End Try
```

كيف نقوم بإظهار مؤشر تقدم لعملية التحميل؟

إذا كنت تقوم بتحميل عدة ملفات يمكنك وضع max للبروجرس بعدد الملفات وكلما تم تحميل ملف نقوم بتقديم المؤشر خطوة أما إذا كنت تريد إظهار مؤشر لعملية التقدم للملف ذات نفسه فمن أجل التحميل استبدل السطر

```
My.Computer.Network.DownloadFile("\\Server\test.txt", " C:\Download\test.txt")
```

بالسطر

```
My.Computer.Network.DownloadFile("\\Server\test.txt", " C:\Download\test.txt", _
                                 "", "", True, 500, True)
```

ومن أجل الرفع استبدل السطر

```
My.Computer.Network.UploadFile("C:\Upload\test.txt", " \\Server\test.txt")
```

بالسطر

```
My.Computer.Network.UploadFile("C:\Upload\test.txt", " \\Server\test.txt", _
                               "", "", True, 500)
```

منع تغيير الصفحة الافتراضية للإنترنت إكسبلورر و تغييرها برمجيا

نستخدم الإجراء التالي لتمكين/عدم تمكين المستخدم من تغيير الصفحة الافتراضية للمتصفح

```
Private Sub EnableDisableIEHomePageChange(ByVal EnHpCh As Boolean)
  Try
    Dim Key As String
    Key = "HKEY_CURRENT_USER\Software\Policies\Microsoft\" _
           & "Internet Explorer\Control Panel"

    Dim Value As String = "HomePage"
    If EnHpCh = True Then
      ' التغيير يمكن
      My.Computer.Registry.SetValue(Key, Value, 0, _
                            Microsoft.Win32.RegistryValueKind.DWord)
    Else
      ' التغيير لايمكن
      My.Computer.Registry.SetValue(Key, Value, 1, _
                        Microsoft.Win32.RegistryValueKind.DWord)
    End If
  Catch ex As Exception
    MsgBox(ex.Message)
  End Try
End Sub
```

وذلك بوضع قيمة الوسيطة EnHpCh الممررة للإجراء إلى False لإيقاف إمكانية تغيير الصفحة الافتراضية و True لتفعيل إمكانية تغيير الصفحة الافتراضية كما في المثال

```
' لإيقاف إمكانية تغيير الصفحة الافتراضية
EnableDisableIEHomePageChange(False)

' لتفعيل إمكانية تغيير الصفحة الافتراضية
EnableDisableIEHomePageChange(True)
```

كما يمكن تغيير الصفحة الافتراضية للمستكشف برمجيا باستخدام الإجراء التالي

```
Private Sub ChangeIEHomePage(ByVal NewUrl As String)
  Try
    Dim Key As String = "HKEY_CURRENT_USER\Software\Microsoft\" _
                     & "Internet Explorer\Main"

    Dim Val As String = "Start Page"
    My.Computer.Registry.SetValue(Key, Val, NewUrl, _
                   Microsoft.Win32.RegistryValueKind.String)
  Catch ex As Exception
    MsgBox(ex.Message)
```

```
  End Try
End Sub
```

مثال: لجعل ياهو هو الصفحة الافتراضية للمتصفح

```
ChangeIEHomePage("www.yahoo.com")
```

القسم السادس - برمجة UAC الخاص بويندوز فيستا

ويضم المواضيع التالية:

- UAC Security
- تمكين برنامجك من استخدام صلاحيات مدير على فيستا
- كيف نقوم بجعل أحد الأزرار في برنامجنا ينفذ أوامر تتطلب صلاحيات مدير في ويندوز فيستا

UAC Security

استعراض UAC

بشكل عام لا يمكن للبرنامج تأدية أعمال تتطلب صلاحيات لا يمتلكها المستخدم فإن لم يمتلك ذلك المستخدم الصلاحيات الكافية لحذف ملفات في مجلد الويندوز فلا يمكن للبرنامج المشغل من قبله أن يحذف تلك الملفات أيضا ومع ذلك يمكن للمستخدم تنفيذ أعمال من المفترض أنه ممنوع منها. والمطورون يعلمون منذ مدة طويلة أن التطبيق يجب أن يمتلك بعض الصلاحيات لكي يتمكن من إتمام العمل فإن كان التطبيق يتطلب العديد من الصلاحيات فوحدهم المستخدمون الذين يمتلكون تلك الصلاحيات يمكنهم تشغيل ذلك البرنامج. ولسوء الحظ فإن العديد من التطبيقات التي تقوم بأعمال قوية تحتاج إلى إنشاء أو حذف ملفات في مجلد الويندوز أو الوصول إلى مناطق متعلقة بالنظام أو التعديل على متغيرات البيئة أو مسجل النظام فإن كان التطبيق يحتاج تلك الصلاحيات فعندها يجب أن يمتلك تلك الصلاحيات عند تشغيله مما يعني أنه على العديد من المستخدمين امتلاك صلاحيات مدير نظام حتى يستطيعوا تشغيل تلك البرامج.

والتعامل مع صلاحيات بهذا المستوى يأتي مع أخطار إضافية فإن أساء التطبيق التصرف فقد يتسبب بانهيار النظام حتى لو كان التطبيق ذات نفسه يعمل بصورة طبيعية فقد يقوم المستخدم بعمل شئ كارثي عن طريق الخطأ عندما يكون قد دخل بصلاحيات مدير فقد يقوم بحذف ملفات هامة يصبح معها من المستحيل استعادة النظام ويكون الحل الأمثل في هذه الحالة هو السماح للتطبيق برفع مستوى الصلاحيات التي يستخدمها بشكل مؤقت أثناء تأديته لتلك الوظائف القوية فإن اخطأ التطبيق عند تشغيله لجزئية معينة من الكود فلن يكون لديه الصلاحيات الكافية لعمل ضرر كبير ولن يكون للمستخدم صلاحيات مدير بشكل دائم وبهذا نكون قد قللنا من احتمال الحوادث المدمرة في النظام.

في نسخ الويندوز السابقة لفيستا عندما تقوم بالدخول كمستخدم يمتلك صلاحيات مدير عندها ستتمكن من القيام بعمل أي شئ تقريبا ولكن في ويندوز فيستا فأن الـ UAC يتصرف بطريقة مختلفة قليلا فعندما تدخل كمدير يكون دخولك عبارة عن شقين الأول عبارة عن مستخدم عادي ذو صلاحيات محدودة والثاني مدير نظام يمتلك كافة الصلاحيات ففي البداية يكون عملك كمستخدم عادي حيث يتم استخدام الشق الثاني عند الحاجة فقط فعندما تريد القيام بعملية تحتاج إلى صلاحيات إضافية فالـ UAC يظهر لك صندوق حوار يسألك الموافقة فإن وافقت على تنفيذ ذلك العمل يتم رفع صلاحياتك بشكل مؤقت إلى مستوى مدير حتى ينتهي تنفيذ ذلك العمل وعندها تعود صلاحياتك إلى مستخدم عادي ثانية وإن كنت قد دخلت باسم مستخدم عادي لا يمتلك صلاحيات مدير فمازال بإمكانك تنفيذ أمر يتطلب تلك الصلاحيات المرتفعة حيث يظهر لك الـ UAC صندوق حوار تحذيري يمكنك من الدخول كمدير فإن قمت بالدخول كمدير بنجاح عندها يتم منحك صلاحيات مدير بشكل مؤقت حتى ينتهي تنفيذ ذلك العمل. ويكون الفرق بين الحالتين بسيطا فعندما تدخل كمدير فإن الـ UAC يسألك موافقتك على العمل بالصلاحيات المرتفعة وإن دخلت كمستخدم آخر فإن الـUAC يطلب منك إدخال كلمة السر الخاصة بالمدير

التصميم من اجل UAC

لن يقوم الـ UAC برفع صلاحيات التطبيق بعد أن تم تشغيله فهو يقوم بإسناد الصلاحيات لذلك التطبيق عندما يبدأ ولن يقوم بعدها بتغيير تلك الصلاحيات فإن احتاج التطبيق للعمل بصلاحيات مرتفعة فعليه أن يحصل على تلك الصلاحيات عندما يبدأ ولتجنب إعطاء التطبيق صلاحيات أكثر من اللازم يجب عليك تقسيم كودك إلى أجزاء بحسب احتياجه لتلك الصلاحيات فالبرنامج الرئيسي يجب أن يعمل بصلاحيات عادية ولاحقا يجب عليه تنفيذ تطبيقات أخرى تعمل بصلاحيات مرتفعة عند الحاجة.

فمثلا إن كان لدينا تطبيق يقوم بحفظ البيانات في قاعدة بيانات Sql Server فهو لا يحتاج لصلاحيات مدير ومع ذلك إن أراد إنشاء ملف بملخص العمليات في مجلد الويندوز – مجلد محمي – فسيحتاج عندها لصلاحيات مدير فيمكنك عندها تقسيم التطبيق إلى عدة أجزاء فالتطبيق الرئيسي يقوم بمعظم العمل وتطبيق آخر يقوم بكتابة معلومات الخلاصة إلى ذلك الملف عندها يمكن للتطبيق الأول تشغيل الثاني من أجل كتابة المعلومات في ذلك الملف.

وعندما يكون بالإمكان يفضل أن تعيد كتابة التطبيق لتجنب استخدام صلاحيات مرتفعة فالعديد من البرامج على سبيل المثال تكون منصبة في المجلد Program Files وهذا من المجلدات المحمية وبهذا إن احتاج التطبيق إلى تخزين معلومات في ملف متواجد بنفس المجلد الذي يحتوي على الملف التنفيذي للتطبيق فسوف يحتاج إلى صلاحيات إضافية للقيام بتلك العملية ويمكنك تجاوز هذه المشكلة بجعل التطبيق يكتب ذلك الملف في المجلد الخاص بالمستخدم الحالي. والعمليات الأخرى التي تحتاج لصلاحيات مرتفعة تتضمن الكتابة إلى المجلدات المحمية والتعامل بشكل مباشر مع العتاد وتعديل الأقسام المحمية في سجل النظام مثل HKEY_LOCAL_MACHINE.

وتقسيم التطبيق إلى أقسام تتطلب صلاحيات مرتفعة وأخرى لا تتطلب تلك الصلاحيات لا يمكن التطبيق من استخدام أقل الصلاحيات الممكنة فحسب ولكنه يبسط القسم الأخطر في كودك ويجعله أسهل للتنقيح فمثلا يمكننا استخدام كود شبيه بالتالي لتنفيذ تطبيق يتطلب صلاحيات مرتفعة

```
Private Sub btnRun_Click() Handles btnRun.Click
   Try
       ' Start the process.
      Dim pro As System.Diagnostics.Process
      pro = System.Diagnostics.Process.Start( _
         txtProgram.Text, txtArguments.Text)
       ' Wait for the process to exit.
      pro.WaitForExit()
       ' Display the process's exit code.
      MessageBox.Show("Exit code: " & pro.ExitCode)
   Catch ex As System.ComponentModel.Win32Exception
       ' This happens if the user fails to elevate to Administrator.
      MessageBox.Show("Operation canceled", _
          "Canceled", MessageBoxButtons.OK, _
         MessageBoxIcon.Information)
   End Try
End Sub
```

الكود السابق يستخدم الوظيفة System.Diagnostics.Process.Start لتشغيل التطبيق ممررا مسار التطبيق الذي نريد تنفيذه ومحددات سطر الأوامر الخاصة بها وهو يستخدم الدالة WaitForExit من الغرض المعاد التي تنتظر حتى الانتهاء من تنفيذ البرنامج ثم يتم التأكد عبر الخاصية ExitCode من القيمة المعادة من التطبيق المنفذ.

ويمثل الكود التالي الإجراء main في البرنامج المستدعى

```
Function Main(ByVal cmdArgs() As String) As Integer
   Dim frm As New frmChoices
    ' Display the arguments.
   For Each str As String In cmdArgs
      frm.lstArguments.Items.Add(str)
   Next str
    ' Select the first item.
   If frm.lstArguments.Items.Count > 0 Then
      frm.lstArguments.SelectedIndex = 0
   End If
    ' Return the index of the selected item.
   If frm.ShowDialog() = DialogResult.Cancel Then
      Return -1
   Else
      Return frm.lstArguments.SelectedIndex
   End If
End Function
```

حيث يبدأ التطبيق بإنشاء نموذج frmChoices وإضافة محددات سطر الأوامر إلى صندوق القائمة lstArguments ونختار العنصر الأول منه ثم نظهر النموذج فإن قام المستخدم بالضغط على الزر Cancel فالتطبيق يعيد القيمة -1 وإن ضغط على الزر OK فهو يعيد قيمة الخاصية Index من صندوق القائمة والموافقة للعنصر الذي تم اختياره منها والكود المستدعي للتطبيق يستقبل تلك القيمة عبر الخاصية ExitCode.

وكجزء من خصائص المستخدم لـ UAC فأي عمل يتطلب صلاحيات مرتفعة يجب أن يتم تعليمه بواسطة الدرع القياسي لـ UAC حيث يجب إظهاره لتحذير المستخدم بأنه ينفذ تطبيق يتطلب صلاحيات مرتفعة. وفي الوقت الحالي لا توجد طريقة بسيطة لإظهار ذلك الدرع في فيجول بايزيك لذلك سنستخدم دالات API لجعل الزر يظهر ذلك الدرع كما هو ظاهر في قطعة الكود التالية

```
Imports System.Runtime.InteropServices

Module UacStuff
    Declare Auto Function SendMessage Lib "user32.dll" _
        (ByVal hWnd As HandleRef, ByVal msg As Int32, _
        ByVal wParam As IntPtr, ByVal lParam As IntPtr) As Int32

    ' Make the button display the UAC shield.
    Public Sub AddShieldToButton(ByVal btn As Button)
        Const BCM_SETSHIELD As Int32 = & H160C
        btn.FlatStyle = Windows.Forms.FlatStyle.System
        SendMessage(New HandleRef(btn, btn.Handle), _
        BCM_SETSHIELD, IntPtr.Zero, CType(1, IntPtr))
    End Sub
End Module
```

ففي البداية نقوم بتعريف الدالة SendMessage المتواجدة في المكتبة User32.dll حيث يقوم الإجراء AddShieldToButton بضبط الخاصية FlatStyle الخاصة بالزر إلى System ثم يستخدم الدالة SendMessage لإرسال الرسالة BCM_SETSHIELD إلى الزر ولا توفر لنا مايكروسوفت حاليا طريقة لإضافة الدرع لتحكمات أخرى غير زر الأوامر فإن أردت إضافته إلى تحكم أخر فستقوم بذلك لوحدك كما يمكنك عمل صورة للدرع ووضعها ببساطة على تحكماتك ولكن هذه الصورة لن تتغير إن تم تغيير صورة الدرع الخاصة بالنظام

رفع صلاحيات البرامج

يمكن للمستخدم رفع المستوى الذي يتم تنفيذ التطبيق ضمنه بواسطة اختيار الأمر Run As Administrator من القائمة التي تظهر لك عند الضغط بزر الفأرة اليميني على الملف التنفيذي للتطبيق فيقوم النظام بإظهار صندوق حوار UAC الخاص وبعد أن يقوم المستخدم بإدخال كلمة سر المدير يتم تنفيذ البرنامج باستخدام الصلاحيات المرتفعة وهذه الطريقة بسيطة ولا تتطلب تدخلا منك كمبرمج ولكنها تتطلب من المستخدم القيام بخطوة إضافية ولهذا قد لا تكون هذه الفكرة هي الحل الأفضل دوما.

ويمكننا جعل تطبيقنا يبدأ تطبيق معين باستخدام صلاحيات مرتفعة بطريقة تشابه تلك الطريقة التي يستخدمها المستخدم فهو يبدأ تشغيل التطبيق طالبا من النظام تشغيله بصلاحيات مدير حيث يمكن استخدام كود شبيه بالتالي لتشغيل تطبيق آخر بصلاحيات مرتفعة مع أن ذلك التطبيق بذاته لا يطلب تلك الصلاحيات عند بدء تشغيله

```
Try
    ' Use the runas verb to start the process.
    Dim psi As New ProcessStartInfo
    psi.Verb = "runas"
    psi.UseShellExecute = True
    psi.FileName = txtProgram.Text
    psi.Arguments = txtArguments.Text
    Dim pro As System.Diagnostics.Process
    pro = System.Diagnostics.Process.Start(psi)
    ' Wait for the process to exit.
    pro.WaitForExit()
    ' Display the process's exit code.
    MessageBox.Show("Exit code: " & pro.ExitCode)
Catch ex As System.ComponentModel.Win32Exception
    ' This happens if the user fails to elevate to Administrator.
    MessageBox.Show("Operation canceled", _
        "Canceled", MessageBoxButtons.OK, _
        MessageBoxIcon.Information)
End Try
```

حيث يبني الكود السابق الغرض ProcessStartInfo واصفا التطبيق الذي سيشغله الكود حيث يقوم بضبط الخاصية Verb إلى القيمة runas لكي يبين للنظام أن التطبيق يجب أن يتم تشغيله كمدير كما يضبط اسم البرنامج ومحددات بدء التشغيل الخاصة به.

وإن كنت تعلم أن التطبيق يجب أن يتم تشغيله دوما باستخدام صلاحيات مرتفعة يمكنك جعل ذلك التطبيق يطلب رفع صلاحياته بنفسه وذلك باستخدام manifest مضمنة داخل الملف التنفيذي للتطبيق ولإنشائها انقر نقرا مزدوجا على My Project في Solution Explorer وفي صفحة Application انقر على الزر View UAC Settings الذي يقوم بفتح الملف app.manifest حيث يظهر الكود التالي المحتويات الابتدائية لذلك الملف

```xml
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"
xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-
com:asm.v2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
            If you want to change the Windows User Account Control level replace the
            requestedExecutionLevel node with one of the following.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

            If you want to utilize File and Registry Virtualization for backward
            compatibility then delete the requestedExecutionLevel node.
        -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</asmv1:assembly>
```

ولجعل التطبيق يطلب من UAC رفع صلاحياته غير قيمة السطر requestExecutionLevel إلى requireAdministrator والآن عندما تقوم بعمل Compile للتطبيق يقوم فيجول ستوديو بتعليم التطبيق بأنه بحاجة إلى صلاحيات مدير فعندما يقوم المستخدم أو أي برنامج آخر بتشغيله سيحاول النظام بصورة آلية رفع صلاحياته مظهرا صندوق الحوار الخاص بـ UAC للمستخدم طالبا منه الموافقة على رفع تلك الصلاحيات

الخلاصة

القواعد الأساسية لبرمجة UAC تتطلب استخدام الحد الأدنى من الصلاحيات لتنفيذ العمل المراد ويجب على التطبيق استخدام صلاحيات مستخدم عادي عندما يكون ذلك ممكنا وإن كان عليه تنفيذ مهمة تتطلب صلاحيات أكبر فيجب عليه تنفيذ تطبيق آخر منفصل يمتلك تلك الصلاحيات المرتفعة.

وقد ورد في هذه المقالة ثلاثة طرق لبدء البرنامج بصلاحيات مرتفعة: الأولى هي الطلب من المستخدم فعل ذلك وذلك من خلال النقر بزر الفأرة اليميني على التطبيق واختيار الأمر Run As Administrator وهذه ليست بالطريقة الملائمة بشكل عام ولكنها تبقى مقبولة إن كان المستخدم سيشغل ذلك التطبيق مرات نادرة والثانية هي جعل التطبيق يبدأ التطبيق الآخر بصلاحيات مرتفعة وهذه طريقة أفضل من الأولى ولكنه مازال بالإمكان تشغيل التطبيق بدون الصلاحيات التي يحتاجها والثالثة هي تضمين manifest ضمن التطبيق المستدعى لجعله يطلب صلاحيات مرتفعة في كل مرة يبدأ فيها تشغيله

تمكين برنامجك من استخدام صلاحيات مدير على فيستا

- لتمكين برنامجك من العمل بصلاحيات مدير شغل بيئة التطوير دوما بصلاحيات مدير - انقر بزر الفأرة اليميني على اختصار بيئة التطوير واختر الأمر Run As Administrator -
- افتح خصائص My Project ثم انقر زر View UAC Settings من صفحة Application ثم في نافذة خصائص UAC التي تظهر لك استبدل السطر

```
<requestedExecutionLevel level="asInvoker" uiAccess="false" />
```

- بالسطر

```
<requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
```

- نفذ الأمر Build Solution من قائمة Build وبيئة التطوير مازالت تعمل ضمن مستوى Administrator كما تأكد بأنك تستخدم بيئة التطوير بصلاحيات مدير عندما تقوم بعمل برنامج الـ Setup أيضا

كيف نقوم بجعل أحد الأزرار في برنامجنا ينفذ أوامر تتطلب صلاحيات مدير في ويندوز فيستا

نحتاج في بعض الأحيان للقيام بأعمال تتطلب صلاحيات خاصة في ويندوز فيستا وهنا سنواجه منعا من قبل UAC الخاص بويندوز فيستا ولكي يتمكن برنامجنا من تنفيذ هذه المهمة يحب علينا تنفيذ ذلك الكود بمستوى صلاحيات مدير Administrator حيث سنقوم في البداية بتعريف فئة تتعامل مع نظام الأمان في ويندوز فيستا مستخدمين الفئة WindowsIdentity للتعرف على مستخدم ويندوز الذي نعمل عليه والفئة WindowsPrincipal للتعرف على المجموعات التي ينتسب إليها ذلك المستخدم ثم نتحقق من أنه يعمل بصلاحيات مدير كما في الإجراء

```
Friend Shared Function IsAdmin() As Boolean
    Dim id As WindowsIdentity = WindowsIdentity.GetCurrent()
    Dim p As WindowsPrincipal = New WindowsPrincipal(id)
    Return p.IsInRole(WindowsBuiltInRole.Administrator)
End Function
```

فإن لم يكن المستخدم يعمل بصلاحيات مدير هنا نعيد بدء العملية الحالية Restart Current Process رافعين مستوى صلاحيات المستخدم إلى مستوى مدير كما في الإجراء

```
Friend Shared Sub RestartElevated()
    Dim startInfo As ProcessStartInfo = New ProcessStartInfo()
    startInfo.UseShellExecute = True
    startInfo.WorkingDirectory = Environment.CurrentDirectory
    startInfo.FileName = Application.ExecutablePath
    startInfo.Verb = "runas"
    Try
        Dim p As Process = Process.Start(startInfo)
    Catch ex As System.ComponentModel.Win32Exception
        Return 'If cancelled, do nothing
    End Try

    Application.Exit()
End Sub
```

بقي لدينا إضافة أيقونة الدرع الخاصة بالأزرار التي تستخدم صلاحيات مدير إلى زر الأوامر المطلوب حيث يتم ذلك باستخدام الدالة SendMessage الموجودة في المكتبة user32.dll التي تقوم بإرسال الرسالة BCM_SETSHIELD إلى الزر المطلوب كما في الإجراء

```
Friend Shared Sub AddShieldToButton(ByVal b As Button)
    b.FlatStyle = FlatStyle.System
    SendMessage(b.Handle, BCM_SETSHIELD, 0, &HFFFFFFFF)
End Sub
```

وسيصبح الكود الكامل للفئة التي سنستخدمها لإجراء عملية تمكين الزر من تنفيذ أعمال تتطلب صلاحيات مدير كما يلي

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Runtime.InteropServices
Imports System.Diagnostics
Imports System.Windows.Forms
Imports System.Security.Principal

Public Class VistaSecurity
    Private Declare Auto Function SendMessage Lib "user32.dll" _
        (ByVal HWND As IntPtr, ByVal MSG As UInteger, ByVal WParam As UInt32, _
         ByVal LParam As UInt32) As UInt32

    Private Const BCM_FIRST = &H1600
    Private Const BCM_SETSHIELD = (BCM_FIRST + &HC)

    Friend Shared Function IsVistaOrHigher() As Boolean
```

```
        Return Environment.OSVersion.Version.Major < 6
    End Function

    '/ <summary>
    '/ Checks if the process is elevated
    '/ </summary>
    '/ <returns>If is elevated</returns>
    Friend Shared Function IsAdmin() As Boolean
        Dim id As WindowsIdentity = WindowsIdentity.GetCurrent()
        Dim p As WindowsPrincipal = New WindowsPrincipal(id)
        Return p.IsInRole(WindowsBuiltInRole.Administrator)
    End Function

    '/ <summary>
    '/ Add a shield icon to a button
    '/ </summary>
    '/ <param name="b">The button</param>
    Friend Shared Sub AddShieldToButton(ByVal b As Button)
        b.FlatStyle = FlatStyle.System
        SendMessage(b.Handle, BCM_SETSHIELD, 0, &HFFFFFFFF)
    End Sub

    '/ <summary>
    '/ Restart the current process with administrator credentials
    '/ </summary>
    Friend Shared Sub RestartElevated()
        Dim startInfo As ProcessStartInfo = New ProcessStartInfo()
        startInfo.UseShellExecute = True
        startInfo.WorkingDirectory = Environment.CurrentDirectory
        startInfo.FileName = Application.ExecutablePath
        startInfo.Verb = "runas"
        Try
            Dim p As Process = Process.Start(startInfo)
        Catch ex As System.ComponentModel.Win32Exception
            Return 'If cancelled, do nothing
        End Try

        Application.Exit()
    End Sub
End Class
```

دعنا نجرب معا الفئة التي قمنا بإنشائها للتو

سأقوم بالتجربة على كود مقتطف من برنامج قديم لي وهو يراقب خدمة النظام الخاصة بـ SQl Server ويتحكم بها وبما أن كود بدء أو إيقاف هذه الخدمة يعتبر من الأمور التي تحتاج إلى صلاحيات مدير لذا سأضع فقط قطعة الكود التي تفيدنا هنا حيث سنحتاج في البداية إلى إضافة مرجع إلى System.ServiceProcess وإلى الاستيرادات التالية في بداية الملف أيضا

```
Imports System.ServiceProcess
Imports Microsoft.Win32
```

وهذا هو الكود

```
Private SqlServiceCon As New _
        System.ServiceProcess.ServiceController("MSSQL$SQLEXPRESS")

Private Sub StopSQL()
    Try
        SqlServiceCon.Refresh()
        If SqlServiceCon.CanStop = True Then SqlServiceCon.Stop()

    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
```

```
End Sub

Private Sub StartSql()
    Try
        SqlServiceCon.Refresh()
        If SqlServiceCon.Status <> ServiceControllerStatus.Running And _
           SqlServiceCon.Status <> ServiceControllerStatus.StartPending Then

            SqlServiceCon.Start()
        End If
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

الآن سنستخدم فئتنا السابقة VistaSecurity للتحقق أولا من أن برنامجنا يمتلك الصلاحيات المطلوبة باستخدام الدالة IsAdmin فإن لم يمتلك تلك الصلاحيات نعيد بدء العملية Process الحالية رافعين الصلاحيات للمستوى المطلوب باستخدام الدالة RestartElevated كما في الكود

```
If VistaSecurity.IsAdmin = True Then
    StartSql()
Else
    VistaSecurity.RestartElevated()
End If
```

وعملية إضافة أيقونة الدرع إلى زر الأوامر تتم باستخدام الكود

```
VistaSecurity.AddShieldToButton(Button1)
```

وفيما يلي سرد كامل لكود النافذة Form1 التي استخدمناها هنا وهي تمتلك زري أوامر Button1 و Button2

```
Imports System.ServiceProcess
Imports Microsoft.Win32

Public Class Form1

    Private SqlServiceCon As New
System.ServiceProcess.ServiceController("MSSQL$SQLEXPRESS")

    Private Sub StopSQL()
        Try
            SqlServiceCon.Refresh()
            If SqlServiceCon.CanStop = True Then SqlServiceCon.Stop()

        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub

    Private Sub StartSql()
        Try
            SqlServiceCon.Refresh()
            If SqlServiceCon.Status <> ServiceControllerStatus.Running And _
               SqlServiceCon.Status <> ServiceControllerStatus.StartPending Then

                SqlServiceCon.Start()
            End If
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button1.Click

        If VistaSecurity.IsAdmin = True Then
            StartSql()
        Else
            VistaSecurity.RestartElevated()
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles Button2.Click

        If VistaSecurity.IsAdmin = True Then
            StopSQL()
        Else
            VistaSecurity.RestartElevated()
        End If
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, _
                ByVal e As System.EventArgs) Handles Me.Load

        VistaSecurity.AddShieldToButton(Button1)
        VistaSecurity.AddShieldToButton(Button2)
    End Sub

End Class
```

سؤال

هل من الممكن شرح استخدام هذه الطريقة مع Windows XP

الجواب

هذه الطريقة خاصة حصرا لـ Windows Vista ولا يمكن استعمالها مع Windows XP

القسم السابع – Linq

ويضم المواضيع التالية:

- مقدمة في Linq
- بنية استعلامات Linq
- مزودات Linq – Linq Providers
- بنية استعلامات Linq
- ترقية مشاريع 2005 لتعمل على 2008 ثم إضافة دعم Linq لتلك المشاريع
- Linq To Object وأساسيات استعلامات Linq
- Linq To DataSet
- مثال عملي على Linq To DataSet مع استخدام Lambda Expressions
- مقدمة في Linq to XML
- بعض استخدامات Linq TO XML
- تعرف على Linq to SQL و O/R Designer
- Linq To Sql Master/Detail
- مثال سريع عن كيفية إنشاء فئات Linq To SQL يدويا
- أمثلة على استعلامات Linq
- الاستعلامات المترجمة Compiled Queries
- إضافة طرائق مخصصة لاستعلامات لينك Linq

مواضيع متعلقة بتقنية Linq لابد من الاطلاع عليها

هذه أسماء مجموعة من المواضيع ستجدها في أقسام هذا الكتاب وهي متعلقة بتقنية Linq يفضل ان تطلع عليها أولا ثم العودة لمتابعة القسم المتعلق بتقنية Linq وهي

- الاستدلال المحلي على النوع Local Type Inference
- الأنواع المجهولة Anonymous Types
- Lambda Expressions
- تعابير لمدا في العمق
- Object Initializers

مقدمة في Linq

تضيف Linq إمكانيات استعلامية بإمكانيات بسيطة وقوية لفيجول بايزيك عندما تتعامل مع العديد من أنواع البيانات المختلفة فبالإضافة إلى إرسال الاستعلام إلى قاعدة بيانات كي تتم معالجته أو العمل مع صيغة مختلفة للاستعلام لكل نوع من أنواع البيانات التي تقوم بالبحث عنها. تقدم Linq الاستعلامات كجزء من لغة فيجول بايزيك مستخدمة صيغة موحدة بغض النظر عن نوع البيانات الذي تستخدمه. وهي تمكنك من الاستعلام عن البيانات من قاعدة بيانات SQL Server أو Xml أو المجموعات والمصفوفات في الذاكرة أو ADO .net Datasets الأمر الذي يجعلها قادرة على الاستعلام من أي قاعدة بيانات يمكن ربطها مع DataSet أو أي مصدر بيانات محلي أو بعيد يدعم Linq حيث يمكنك عمل ذلك كله باستخدام عناصر لغة فيجول بايزيك الشائعة لأن استعلاماتك أصبحت مكتوبة بلغة فيجول بايزيك ونتائج الاستعلام تعاد كأغراض أنواع بيانات قوية داعمة IntelliSense مما يجعل كتابتك للكود أسرع واكتشافك للأخطاء في الاستعلامات عند ترجمة المشروع بدلا من وقت التنفيذ كما أن استعلامات Linq يمكن أن تكون مصدرا لاستعلامات إضافية لمزيد من الدقة في البحث. كما يمكن ربطها مع التحكمات ممكنا المستخدم من استعراض وتعديل نتائج استعلامك بسهولة.

وهذا مثال عن استعلام بسيط يعيد قائمة شركات الزبائن الموجودين في إيطاليا

```
Dim itaCus = From cus In NwDs.Customers _
             Where cus.Country = "Italy" _
             Select cus.ContactName, cus.CompanyName
```

دعنا لا نقلق الآن بخصوص صيغة الاستعلام على كل حال إن كنت متمكنا من كتابة استعلامات سيكول سيرفر لن تجد صعوبة في فهم صيغتها بما أن الصيغة مشابهة مع بعض الاختلافات طبعا والاستعلام السابق يماثل استعلام سيكول سيرفر التالي

SELECT COMPANYNAME FROM CUSTOMERS WHERE COUNTRY='Italy'

كما يمكن أن يكون استعلام Linq أكثر تعقيد فالكود التالي يعيد قائمة بالزبائن ويعيد تجميعهم حسب الموقع

```
Dim customers As List(Of Customer) = GetCustomerList()
Dim customersByCountry = From cust In customers _
                         Order By cust.Country, cust.City _
                         Group By CountryName = cust.Country _
                         Into RegionalCustomers = Group, Count() _
                         Order By CountryName
For Each country In customersByCountry
     Console.WriteLine(country.CountryName & _
           " (" & country.Count & ")" & vbCrLf)
    For Each customer In country.RegionalCustomers
         Console.WriteLine(vbTab & customer.CompanyName & _
               " (" & customer.City & ")")
    Next
Next
```

حيث يمكننا استخدام استعلام Linq في برنامجنا بعدة أشكال فالمثال الأول مثلا يمكننا عرض نتيجته في DataGrid مباشرة

```
Me.DataGridView1.DataSource = itaCus.ToList
```

أو يمكننا إدخاله ضمن حلقة For...Each مثلا واستخدام النتائج ففي المثال التالي نستخدم الاستعلام الوارد بالمثال الأول لإظهار قائمة الشركات في ListBox

```
Me.ListBox1.Items.Clear()
For Each cust In itaCus
       Me.ListBox1.Items.Add(cust.CompanyName)
Next
```

LINQ Providers - مزودات Linq

يقوم مزود Linq بتنظيم استعلامات Linq في فيجول بايزيك بحسب مصدر البيانات الذي تستخدمه فعندما تكتب استعلام Linq يأخذ المزود ذلك الاستعلام ويترجمه إلى أوامر يستطيع مصدر البيانات تنفيذها ويقوم أيضا بتحويل البيانات من الأغراض المصدرية ليشكل نتائج الاستعلام وأخيرا يقوم بتحويل الأغراض إلى بيانات عندما ترسل التحديثات للمصدر. ويضم فيجول بايزيك مزودات Linq التالية:

- Linq to Objects يمكنك هذا المزود من الاستعلام في المجموعات والمصفوفات في الذاكرة إذا كانت غرضك يدعم الواجهة IEnumerable أو الواجهة IEnumerable(T) بحيث يمكنك المزود من الاستعلام عنها ويمكنك تمكين هذا المزود باستيراد المجال System.Linq والذي يكون مستوردا بشكل افتراضي في مشاريع فيجول بايزيك
- Linq to SQL يمكنك هذا المزود من الاستعلام من قواعد بيانات SQL Server والتحديث إليها ويجعل من السهل ربط أغراض التطبيق مع الجداول والأغراض في قواعد البيانات. ويسهل فيجول بايزيك العمل مع Linq to SQL بتقديم Object Relational Designer والذي يمكنك من إنشاء Object Model في التطبيق يرتبط مع الأغراض في قاعدة البيانات ويقدم الـ O/R Designer إمكانية التعامل مع الإجراءات والوظائف المخزنة وإجراءات الغرض DataContext الذي يدير الاتصال مع قاعدة البيانات ويخزن الحالة من أجل تصادم البيانات التفاؤلي
- Linq to Xml يمكنك من الاستعلام من Xml والتعديل عليها بحيث يمكنك تعديل محتويات Xml الموجودة في الذاكرة أو يمكنك تحميل ملف Xml أو حفظه
- Linq to Dataset يمكنك من الاستعلام من ADO .net Datasets والتعديل عليها وإضافة قوة Linq للتطبيقات التي تستخدم Datasets تسهل وتوسع إمكانيات الاستعلام والتجميع والتحديث في الـ Dataset في تطبيقك

بنية استعلامات Linq

يشار عادة إلى استعلام Linq بتعبير الاستعلام وهو يتألف من توليفة من تراكيب الاستعلام التي تحدد مصدر البيانات ومتغيرات التكرار الخاصة بالاستعلام كما يمكنه أن يتضمن تعليمات من أجل الفرز أو التصفية أو التجميع أو الضم أو الحساب ليتم تطبيقها على البيانات المصدرية وصيغتها تكون مشابهة لصيغة الـ SQL ولهذا ستجد أن معظم الصيغة مألوفة.
يبدأ الاستعلام بقسم From الذي يحدد مصدر البيانات والمتغيرات التي تشير إلى كل عنصر من البيانات المصدرية بشكل مستقل وهي تدعى بمتغيرات المجال أو متغيرات التكرار وقسم From مطلوب من أجل الاستعلام إلا في استعلامات التجميع Aggregate حيث يكون قسم From فيها اختياري وبعد تعريف مجال ومصدر الاستعلام في قسم From أو في قسم Aggregate يمكنك تضمين أي تركيب من أقسام الاستعلام. فمثلا الاستعلام التالي يحدد مصدر مجموعة من بيانات الزبائن بالمتغير Customers ومتغير التكرار cust

```
Dim queryResults = From cust In customers _
                   Select cust.CompanyName
```

وهذا المثال يشكل استعلام مقبول بذات نفسه ومع ذلك يصبح الاستعلام أقوى عندما تضيف أقسام استعلام أخرى لتحديد النتائج فمثلا يمكنك إضافة قسم Where لتصفية النتائج إلى قيمة أو أكثر وتكون تعابير الاستعلام عبارة عن سطر واحد من الكود بحيث يمكنك إضافة أقسام استعلام جديدة لنهاية الاستعلام كما يمكنك فصل الاستعلام إلى عدة أسطر لتحسين قراءة كودك باستخدام المحرف _ ويمثل الكود التالي استعلاما يستخدم قسم Where

```
Dim queryResults = From cust In customers _
                   Where cust.Country = "USA"
```

ويمثل قسم select قسم قوي آخر في الاستعلام حيث يمكنك من إعادة الحقول المختارة فقط من مصدر البيانات وتعيد استعلامات Linq مجموعة تعدادية من الأغراض القوية النوع كما يمكنها إعادة أنواع مجهولة أو أنواع معروفة. ويمكن استخدام قسم select للعودة بحقل واحد فقط من مصدر البيانات وعندما تفعل هذا يكون نوع المجموعة المعادة هو نوع بيانات ذلك الحقل. وعندما يعيد قسم Select مجموعة من الحقول من مصدر البيانات تكون المجموعة المعادة من النوع المجهول ويمكنك مطابقة الحقول المعادة من الاستعلام مع حقول من نوع معروف محدد ويظهر الكود التالي تعبير استعلام يعيد مجموعة نوعها مجهول تضم أرقاما مع بيانات من الحقل المحدد من مصدر البيانات

```
Dim queryResults = From cust In customers _
                   Where cust.Country = "USA" _
                   Select cust.CompanyName, cust.Country
```

يمكن استخدام استعلامات Linq لدمج عدة مصادر من البيانات في نتيجة واحدة حيث يمكن عمل هذا باستخدام قسم From واحد أو أكثر أو باستخدام أقسام Join أو Group Join ويظهر الكود التالي تعبير استعلام يضم بيانات Customer و Order ويعيد مجموعة من نوع مجهول تحتوي بيانات من Customer و Order

```
Dim queryResults = From cust In customers, ord In orders _
                   Where cust.CustomerID = ord.CustomerID _
                   Select cust, ord
```

يمكنك استخدام قسم Group Join لبناء استعلامات شجرية تحتوي مجموعة من أغراض Customer وكل غرض Customer يمتلك خاصية تحتوي مجموعة تتضمن جميع أغراض order لذلك الزبون. ويظهر المثال التالي تعبير استعلام يدمج بيانات Customer و Order كنتيجة شجرية ويعيد مجموعة من نوع مجهول ويعيد الاستعلام نوعا يتضمن الخاصية CustomerOrders تحتوي على مجموعة تحتوي على مجموعة من بيانات Order وبيانات Customer وتتضمن أيضا الخاصية OrderTotal تحتوي على مجموع إجمالي الطلبات لذلك الزبون

```
Dim queryResults = From cust In customers _
                   Group Join ord In orders On _
                   cust.CustomerID Equals ord.CustomerID _
                   Into CustomerOrders = Group, _
                        OrderTotal = Sum(ord.Total) _
                   Select cust.CompanyName, cust.CustomerID, _
                          CustomerOrders, OrderTotal
```

معاملات استعلام Visual Basic LINQ Query Operators - Linq

تتضمن الفئات في المجال System.Linq والمجالات التي تدعم Linq طرائق يمكنك استدعاؤها لإنشاء الاستعلامات وتوليها بحسب حاجة التطبيق ويتضمن فيجول بايزيك كلمات مفتاحية لأقسام الاستعلام الشائعة

From Clause

يجب أن يبدأ الاستعلام بقسم From أو Aggregate ويحدد قسم From المجموعة المصدر أو متغير التكرار للاستعلام

```
' Returns the company name for all customers for whom
' State is equal to "WA".
Dim names = From cust In customers _
            Where cust.State = "WA" _
            Select cust.CompanyName
```

Select Clause

اختياري يحدد مجموعة من متغيرات التكرار للاستعلام

```
' Returns the company name and ID value for each
' customer as a collection of a new anonymous type.
Dim customerList = From cust In customers _
                   Select cust.CompanyName, cust.ID
```

و إن لم يكن قسم Select موجودا في الاستعلام فتتألف متغيرات التكرار للاستعلام من تلك المحددة في قسم From أو Aggregate

Where Clause

اختياري ويحدد شرط التصفية للاستعلام

```
' Returns all product names for which the Category of
' the product is "Beverages".
Dim names = From product In products _
            Where product.Category = "Beverages" _
            Select product.Name
```

Order By Clause

اختياري ويحدد ترتيب الفرز للأعمدة في الاستعلام

```
' Returns a list of books sorted by price in
' ascending order.
Dim titlesAscendingPrice = From b In Books _
                           Order By b.Price
```

Join Clause

اختياري ويدمج مجموعتين ضمن مجموعة واحدة

```
Dim Intrst = From i In DsDesposits.InterestRates_
             Join d In DsDesposits.Deposits_
             On i.InterestID Equals d.InterestID_
             Select i.InterestID, i.InterestRate, i.DepositPreiod
```

Group by Clause

اختياري ويقوم بتجميع عناصر نتيجة الاستعلام ويمكن استعماله لتطبيق إجراءات تجميع لكل مجموعة

```
' Returns a list of orders grouped by the order date
' and sorted in ascending order by the order date.
Dim orders = From order In orderList _
             Order By order.OrderDate _
             Group By OrderDate = order.OrderDate _
             Into OrdersByDate = Group
```

Group Join Clause

اختياري ويجمع مجموعتين ضمن مجموعة شجرية واحدة

```
' Returns a combined collection of customers and
' customer orders.
Dim customerList = From cust In customers _
                   Group Join ord In orders On _
                   cust.CustomerID Equals ord.CustomerID _
                   Into CustomerOrders = Group, _
                        OrderTotal = Sum(ord.Total) _
                   Select cust.CompanyName, cust.CustomerID, _
                          CustomerOrders, OrderTotal
```

Aggregate Clause

يجب بدء الاستعلام دوما إما بقسم From أو قسم Aggregate وقسم Aggregate يطبق واحدة أو أكثر من وظائف التجميع على المجموعة فمثلا يمكن استخدام قسم Aggregate لحساب مجموع جميع العناصر المعادة بالاستعلام

```
' Returns the sum of all order totals.
Dim orderTotal = Aggregate order In Orders _
                 Into Sum(order.Total)
```

كما يمكنك استخدام قسم Aggregate لتعديل الاستعلام فمثلا يمكن استخدام قسم Aggregate لإجراء عملية حسابية على مجموعة استعلام

```
' Returns the customer company name and largest
' order total for each customer.
Dim customerMax = From cust In customers _
                  Aggregate order In cust.Orders _
                  Into MaxOrder = Max(order.Total) _
                  Select cust.CompanyName, MaxOrder
```

Let Clause

اختياري ويقوم بحساب قيمة ويضعها في متغير جديد

```
' Returns a list of products with a calculation of
' a ten percent discount.
Dim discountedProducts = From prod In products _
                         Let Discount = prod.UnitPrice * 0.1 _
                         Where Discount >= 50 _
                         Select prod.ProductName, prod.UnitPrice, Discount
```

Distinct Clause

اختياري وهو يضبط القيم المعادة من الاستعلام بحيث لا يجلب قيما مكررة

```
' Returns a list of cities with no duplicate entries.
Dim cities = From item In Customers _
             Select customer.City_
             Distinct
```

Skip Clause

اختياري يتجاوز عددا معينا من العناصر في المجموعة ويعيد الباقي

```
' Returns a list of customers. The first 10 customers
' are ignored and the remaining customers are
' returned.
Dim customerList = From cust In customers _
                   Skip 10
```

Skip While Clause

اختياري يتجاوز عناصر المجموعة طالما قيمة الشرط True ثم يعيد باقي العناصر

```
' Returns a list of customers. The query ignores all
' customers until the first customer for whom
' IsSubscriber returns false. That customer and all
' remaining customers are returned.
Dim customerList = From cust In customers _
                   Skip While IsSubscriber(cust)
```

Take Clause

اختياري ويعيد عددا من العناصر المتجاورة في بداية المجموعة

```
' Returns the first 10 customers.
Dim customerList = From cust In customers _
                   Take 10
```

Take While Clause

اختياري يقوم بتضمين عناصر المجموعة طالما قيمة الشرط True ويتجاهل بقية العناصر

```
' Returns a list of customers. The query returns
' customers until the first customer for whom
' HasOrders returns false. That customer and all
' remaining customers are ignored.
Dim customersWithOrders = From cust In customers _
                          Order By cust.Orders.Count Descending _
                          Take While HasOrders(cust)
```

كما يمكنك استخدام خصائص إضافية لاستعلام Linq باستدعاء عناصر المجموعات والأنواع المستعلم عنها التي يوفرها Linq حيث يمكنك استخدام هذه الإمكانيات الإضافية باستدعاء معامل استعلام على نتيجة الاستعلام فمثلا الكود التالي يستخدم الطريقة Union لدمج ناتج استعلامين في نتيجة استعلام واحدة ويستخدم الطريقة ToList(TSource) لإعادة ناتج الاستعلام كقائمة

```
Public Function GetAllCustomers() As List(Of Customer)
    Dim customers1 = From cust In domesticCustomers
    Dim customers2 = From cust In internationalCustomers

    Dim customerList = customers1.Union(customers2)
    Return customerList.ToList()
End Function
```

ترقية مشاريع 2005 لتعمل على 2008 ثم إضافة دعم Linq لتلك المشاريع

- افتح مشروعك ضمن بيئة تطوير 2008 فيظهر لك معالج الترقية تلقائيا – اضغط next
- يظهر لك المعالج خيار عمل نسخة احتياطية للملفات القديمة فنختار الخيار Yes, create a backup before converting ثم يقترح مكانا لوضع النسخة الاحتياطية فيه في مربع النصوص تحت الكلمة Location for backup حيث يمكنك تغييره بالضغط على الزر Browse أو تركه كما هو - اضغط next فيظهر لك معلومات عن الترقية - اضغط هنا Finish للبدء بعملية الترقية
- بعد الانتهاء تظهر لك نافذة تخبرك بانتهاء عملية الترقية وفيها خيار Show the conversion log when the wizard is closed وبتفعيل هذا الخيار يظهر لك تقرير عن عملية التحويل بعد إغلاق المعالج هنا اضغط Close
- كما تجدر ملاحظة أن مشروعنا بعد التحويل حتى هذه النقطة مازال متوافقا مع بيئة تطوير الـ2005
- من Solution Explorer انقر بزر الفأرة اليساري نقرا مزدوجا على My Project لفتح خصائصه
- افتح صفحة Compile وقم بضبط الخيار Option Infer إلى On الذي يخبر المعالج أن يستدل على نوع المتغيرات المحلية من التعبير الذي يضبط قيمة ذلك المتغير إن لم نزوده بنوع ذلك المتغير - ولمزيد من المعلومات حول هذا الخيار يمكنك قراءة موضوعي في المنتدى بعنوان الاستدلال المحلي على النوع Local Type Inference
- انتقل لأسفل صفحة Compile ثم اضغط الزر Advanced Compile Options الذي يظهر لنا صندوق حوار بخيارات الترجمة المتقدمة حيث نرى في أسفل هذه النافذة الخيار Target framework والذي مازال مضبوطا على الـ Framework 2.0 حتى الآن حيث يمكننا هذا الخيار من كتابة كود يعمل على أي نسخة من نسخ الفريمورك الموجودة ضمن بيئة تطوير واحدة فبدلا من تنصيب عدة نسخ من Visual Studio على نفس الجهاز بهدف العمل مع أكثر من Framework أصبح الآن بإمكانك عمل ذلك من داخل بيئة تطوير واحدة هي 2008 ومن أجل تمكين Linq سنختار Framework 3.5 ثم اضغط على Ok فيظهر لنا تحذير بأنه سيتم إغلاق وفتح المشروع تلقائيا وأنه سيقوم بحفظ أية تغييرات غير محفوظة تلقائيا وهنا اضغط Yes
- انقر بزر الفأرة اليساري نقرا مزدوجا على My Project لفتح خصائصه وانتقل للصفحة References فنلاحظ أنه قد تم إضافة مرجعا تلقائيا للمكتبة system.core.dll من الـ Framework 3.5 للمشروع وذلك لأننا قمنا بترقية Target Framework ومن أجل تمكين استعلامات Linq علينا إضافة بعض المراجع واستيراد بعض مجالات الأسماء اعتمادا على مزود Linq الذي نحتاج لاستخدامه
- فمن أجل إضافة مرجع لـ Linq to Object نختار من القائمة أسفل Import Namespaces المجال System.linq الذي يمكننا من كتابة استعلامات Linq على الأغراض المختلفة حيث أصبح بإمكاننا كتابة الاستعلام التالي للحصول على أسماء الملفات في المجلد الحالي

```
Dim MyFiles = From Files In My.Computer.FileSystem.GetFiles(CurDir()) _
              Select Files
```

ونلاحظ أنه بسبب خاصية Option Infer المضبوطة إلى On أن المترجم قد ضبط نوع المتغير MyFiles إلى IEnumerable(Of String) وذلك عند تمرير مشيرة الفأرة فوق المتغير MyFiles

- ومن أجل تمكين استعلامات Linq للاستعلام من الـ Datasets سنحتاج لبعض الإجراءات الإضافية وهنا افتح خصائص My Project وعد للصفحة References واضغط الزر Add ومن صفحة .net أضف مرجعا لـ System.Data.DataSetExtensions ثم اضغط OK
- سنحتاج الآن لإعادة توليد الـ Dataset التي نريد استخدامها مع استعلامات Linq ولعمل ذلك ننقر بزر الفأرة اليميني على الـ Dataset المناسبة ثم نختار Run Custom Tool من القائمة وهنا تقوم بيئة التطوير بإعادة كتابة كود الـ Dataset لتصبح الجداول ضمنها موروثة من فئة داعمة لـ Linq تسمى TypedTableBase وهذا الذي يستدعي الحاجة لإضافة مجال الأسماء System.Data.DataSetExtensions وأصبح بإمكاننا الآن كتابة استعلامات من الـ dataset مثل الاستعلام

```
Dim MyCats = From category In Me.CategoryProductDataSet.Categories _
             Where category.CategoryName Like "C*" _
             Select category
```

ونلاحظ هنا أيضا أن المترجم قد ضبط نوع المتغير MyCats بناء على جملة الاستعلام المرتبطة به تلقائيا إلى EnumerableRowCollection

- ولكتابة استعلامات Linq to XML عد إلى صفحة References في خصائص MyProject واضغط الزر Add وأضف مرجعا لـ System.Xml.Linq ثم انتقل إلى القائمة Imported namespaces وقم باختيار System.Xml.Linq حتى نتمكن من كتابة استعلامات من XML

- كما يوجد مزود آخر ربما نريد استخدامه وهو مزود Linq to Sql حيث يمكن إضافته بسهولة فقط قم باختيار Add New Item من قائمة Project وقم بإضافة Linq to Sql Classes وهذا سيقوم تلقائيا بإضافة المراجع والاستيرادات المناسبة لمشروعنا حيث سنلاحظ من صفحة References في خصائص MyProject أنه قد تم إضافة مرجعا لـ System.Data.Linq

أصبح الآن بإمكاننا استخدام Linq للاستعلام على الأغراض Objects المختلفة في مشروعنا بالإضافة إلى الاستعلام من Dataset أو XML أو حتى Sql Databse

Linq To Object وأساسيات استعلامات Linq

باستخدام مزود Linq to Object يمكننا الاستعلام من أغراض دوت نيت المختلفة طالما هي تدعم الواجهة IEnumerable أو الواجهة IEnumerable(T) فمثلا يمكننا كتابة استعلام للحصول على قائمة بالملفات الموجودة في المجلد الحالي

```
Dim Files = From Fi In My.Computer.FileSystem.GetFiles(CurDir()) _
            Order By Fi
```

نلاحظ بداية أننا عندما قمنا بالتصريح عن المتغير Files في بداية الاستعلام لم نصرح عن نوع المتغير وذلك لأن المترجم هنا يستدل على نوعه من التعبير الذي يضبط قيمته وهنا أفترض أنك قد درست موضوعي بخصوص الاستدلال المحلي عن النوع وفي حالة الاستعلام السابق إن قمت بتمرير مؤشر الفأرة فوق المتغير Files ستجد أن بيئة التطوير قامت بضبط نوعه إلى System.Linq.IOrderedEnumeable(Of String) باستخدام الاستدلال المحلي على النوع ونبدأ بكتابة الاستعلام بالقسم From حيث نحدد أنه لدينا متغير Fi يحصل على قيمته من الكائن الذي يلي الكلمة In وفي استعلامنا هنا My.Computer.FileSystem.GetFiles(CurDir()) حيث تجدر الملاحظة إلى أن استعلامات Linq جميعها تكون سطرا واحدا في لغة فيجول بايزيك لذا من أجل التنسيق وقابلية القراءة والتعديل نقسم العبارة على عدة أسطر باستخدام محرف المتابعة _ ونريد هنا أن نخرج قائمة مرتبة بأسماء الملفات لذا نستخدم قسم OrderBy ليقوم بترتيب أسماء الملفات المعادة من الاستعلام وكي نظهر قائمة الملفات هذه في ListBox نستخدم حلقة For ... Each للدوران عبر عناصر المجموعة المعادة من الاستعلام بما أنها تحقق الواجهة IEnumerable وإضافتها عنصرا عنصرا لمربع القائمة كما في المثال

```
For Each f In Files
    Me.ListBox1.Items.Add(f)
Next
```

كما يمكننا استخدام الاستعلام ضمن استعلام آخر فللحصول على معلومات الملفات المعادة من الاستعلام السابق يمكننا كتابة الاستعلام التالي حيث يحدد القسم Select أن النتيجة المعادة هي مجموعة من FileInfo بحسب القيمة المعادة من الدالة GetFileInfo

```
Dim FInfo = From File In Files _
            Select My.Computer.FileSystem.GetFileInfo(File)
```

ويمكننا إظهار نتيجة هذا الاستعلام في DataGridView مباشرة وذلك بضبط قيمة الخاصية DataSource إلى نتيجة عائد الدالة ToList الخاصة بمتغير الاستعلام FileInfo كما في الكود

```
Me.DataGridView1.DataSource = FInfo.ToList
```

كما يمكننا استخدام قسم Select لتخصيص المعلومات المعادة من الاستعلام وبشكل مشابه لما كنا نفعله في عبارة Select التي نستخدمها في استعلامات SQL فبدلا من إعادة كافة خصائص FileInfo كما في الاستعلام السابق يمكننا كتابة استعلامنا لإظهار اسم الملف ووقت الإنشاء فقط كما في المثال

```
Dim MyInfo = From Fi In FInfo _
             Select Fi.Name, Fi.CreationTime
```

كما يمكننا إعادة نتيجة هذا الاستعلام ضمن غرض من إنشائنا بدلا من النوع الذي يتم تحديده تلقائيا كنتيجة للاستعلام فإن كان لدينا فئة بسيطة باسم MyFiles تمتلك خاصيتين CreationTime من النوع Date و Name من النوع String يمكننا عندها إعادة كتابة استعلامنا بالشكل

```
Dim MyFiles = From Fi In FInfo _
            Select New MyFiles() With {.Name = Fi.Name, .CreationTime = Fi.CreationTime}
```

حيث سيعيد الاستعلام النتيجة كمجموعة IEnumerable(Of MyFiles) ويمكنك مراجعة موضوعي Object Initializers بخصوص صيغة تعريف الفئة MyFiles ضمن الاستعلام

وإن أردنا تخصيص ناتج الاستعلام للحصول على الملفات التي تحمل الامتداد exe فقط مثلا نستخدم قسم where الذي يحدد شرط الانتقاء في جملة الاستعلام مستخدمين الطريقة EndsWith لاختيار اسم الملف الذي ينتهي بـ exe. عندها يمكننا كتابة الاستعلام بالشكل التالي

```
Dim ExeFiles = From Fi In My.Computer.FileSystem.GetFiles(CurDir()) _
               Where Fi.EndsWith(".exe") _
               Select My.Computer.FileSystem.GetFileInfo(Fi)
```

وإن أردنا الحصول على مجموع حجوم الملفات من النوع exe يمكننا استخدام ناتج الاستعلام السابق في استعلام جديد

```
Dim ExeSize = Aggregate Fs In ExeFiles _
              Into Sum(Fs.Length)

Me.TextBox1.Text = ExeSize
```

فهنا استخدمنا Aggregate بدلا من From في بداية الاستعلام عندما نريد استخدام الدالات التجميعية للحصول على نتائج تجميعية من الاستعلام ففي قسم Into استخدمنا الدالة Sum للحصول على مجموع الحجوم

إذا افترضنا أنه لدينا فئة باسم Personnel تمتلك الخصائص Name و Birthdate و Age و City و Salary وقمنا بإنشاء قائمة تحتوي على مجموعة عناصر تمتلك نوع هذه الفئات

```
Dim Pers As New List(Of Personnel)
```

وبافتراض أن هذه القائمة تحتوي على العديد من العناصر يمكننا كتابة مجموعة من الاستعلامات للحصول على معلومات مختلفة حول عناصر هذه القائمة فإن أردنا قائمة بالأشخاص الذين رواتبهم أكثر من 10000 وأردنا في الناتج فقط الاسم والعمر والراتب وترتيب النتائج بحسب الراتب يمكننا كتابة الاستعلام كما يلي

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary
```

وإن أردنا فقط الأشخاص الذين يقيمون في مدينة دمشق فقط سيصبح استعلامنا بالشكل

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 And p.City = "Damascus" _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary
```

وإذا أردنا الأشخاص في مدينتي دمشق وحماة الذين رواتبهم أكثر من10000 يصبح استعلامنا كما يلي

```
Dim Prs = From p In Pers _
          Where p.Salary > 10000 _
          And (p.City = "Damascus" Or p.City = "Hama") _
          Order By p.Salary _
          Select p.Name, p.Age, p.Salary, p.City
```

وإن أردنا الحصول على مجموع رواتب الأشخاص المقيمين في حلب يمكننا كتابة الاستعلام

```
Dim prs = Aggregate p In Pers _
          Where p.City = "Aleppo" _
          Into Sum(p.Salary)
```

وإن أردنا الحصول على معلومات الشخص الذي يحصل على اعلى راتب

```
Dim pr = From p In Pers _
         Aggregate pa In Pers _
         Into a = Max(pa.Salary) _
         Where p.Salary = a _
         Select p
```

وإن أردنا معلومات من يحصل على أقل راتب في مدينة حلب

```
Dim pr = From p In Pers _
         Aggregate pa In Pers _
         Where pa.City = "Aleppo" _
         Into a = Min(pa.Salary) _
         Where p.Salary = a And p.City = "Aleppo" _
         Select p
```

وبافتراض انه لدينا فئة ثانية باسم Branches تمتلك الخصائص BranchName و City وقمنا بإنشاء قائمة تحتوي على مجموعة عناصر من نوع هذه الفئات

```
Dim Brnch As New List(Of Branches)
```

يمكننا كتابة الاستعلام التالي للحصول على اسم الشخص والفروع المتوفرة في مدينته

```
Dim BrnPer = From pe In Pers _
             Join br In Brnch On pe.City Equals br.City _
             Select PersonName = pe.Name, BranchName = br.BranchName, pe.City
```

حيث استخدمنا join لربط مجموعة الأشخاص مع مجموعة الفروع باستخدام المدينة ثم استخدمنا Select لتحديد الحقول المطلوب إخراجها في نتيجة الاستعلام

الآن نريد إظهار قائمة بجميع الأشخاص مع الفروع المتوفرة لهم وبذلك سيصبح استعلامنا بالشكل

```
Dim PreBr = From pe In Pers _
            Group Join br In Brnch On pe.City Equals br.City _
            Into AvBr = Group _
            Select pe, AvBr
```

في البداية اخترنا المتغير Pe من قائمة الأشخاص Pers ثم استخدمنا Group Join لربط هذه القائمة مع قائمة الفروع باستخدام حقل المدينة ثم وضعنا ناتج الربط من القائمة الثانية في متغير AvBr في قسم Into ثم حددنا في قسم Select النتائج التي نريدها Pe و AvBr

ويمكننا إظهار ناتج الاستعلام في ListBox باستخدام حلقتي For ... Each متداخلتان

```
For Each a In PreBr
    Me.ListBox1.Items.Add(a.pe.Name & ": " & a.pe.City)
    For Each b In a.AvBr
        Me.ListBox1.Items.Add(".... " & b.BranchName)
    Next
Next
```

أو إذا أردنا استخدام DataGridView لإظهار النتائج سنحتاج إلى تحكمان DataGridView و تحكمان BindingSource ولعمل ذلك في البداية سنعرف متغيرا على مستوى النموذج من النوع Dictionary حيث تكون المفاتيح هي الأشخاص والقيم هي الفروع كما يلي

```
Private Gper As Dictionary(Of Personnel, IEnumerable(Of Branches))
```

ثم نقوم بوضع ناتج الاستعلام في متغيرنا Gper حيث نستخدم تعابير لمدا – هل درست المواضيع المتعلقة بتعابير لمدا - لإضافة المفاتيح والقيم إليه كما في الكود

```
Gper = PreBr.ToDictionary(Function(x) x.pe, Function(y) y.AvBr)
```

ثم سنقوم بربط تحكمات BindingSource مع تحكمات DataGridView كما في الكود

```
Me.DataGridView1.DataSource = Me.BindingSource1
Me.DataGridView2.DataSource = Me.BindingSource2
```

ثم نقوم بضبط خاصية DataSource لـ BindingSource1 إلى قيم مفاتيح Gper

```
Me.BindingSource1.DataSource = Gper.Keys
```

وبذلك يتم إظهار قيم المفاتيح في DataGridView1 التي ستظهر قائمة الأشخاص ولإظهار قائمة الفروع المتوفرة لكل شخص في DataGridView1 في DataGridView2 نقوم بمعالجة الحدث CurrentChanged لـ BindingSource1 باستخدام سطر الكود التالي

```
Me.BindingSource2.DataSource = Gper(CType(Me.BindingSource1.Current, Personnel))
```

الذي يحصل على القيمة في الـ Dictionary المقابلة للسجل الحالي في BindingSource1 ويضبطها كـ DataSource لـ BindingSource2 فيتم عرضها في DataGridView2 التي ستظهر قائمة الفروع المتوفرة لذلك الشخص عند النقر عليه في DataGridView1

Linq To DataSet

بافتراض أن قاعدة البيانات Northwind مثبتة في جهازك من نافذة Data Sources في بيئة التطوير أضف مصدر بيانات جديد لمشروعك يتضمن الجدولين Categories و Products وباستخدام طريقة السحب والإفلات اسحب الجدول Categories إلى سطح النافذة الفارغ لإضافة DataGridView مع بعض التحكمات للنموذج ثم من نافذة DataSources وسع العقدة بجانب الجدول Categories واسحب الجدول Products الذي بداخل الجدول Categories إلى سطح النافذة ليتم إنشاء DataGridView ثانية أسفل الأولى خاصة بالجدول Products طبعا لن أقوم بشرح هذه العملية بتفصيل أكثر بما أنها برمجة قواعد بيانات ودورتنا تتحدث عن Linq فإن واجهت مشكلة ابحث في القسم المناسب في المنتدى أو حاول رؤية فيديوهات ميكروسوفت التعليمية بخصوص هذه النقطة وقبل المتابعة يجب أن تتأكد أن مشروعك يعمل جيدا وأن الـ DataGridView الثانية تعرض البيانات المقابلة لما تم اختياره من الأولى فقط.

انتقل إلى محرر الكود للنموذج وقم بإنشاء إجراء معالجة للحدث CurrentChanged للتحكم CategoriesBindingSource وللحصول على السطر الحالي نستخدم الكود التالي

```
Dim row As NorthwindDataSet.CategoriesRow
row = CType(CType(Me.CategoriesBindingSource.Current, _
      DataRowView).Row, NorthwindDataSet.CategoriesRow)
```

وللحصول على إجمالي قيمة البضائع في تلك الفئة والتي مازال إنتاجها مستمرا يمكننا كتابة الاستعلام

```
Dim Total = Aggregate Product In NorthwindDataSet.Products _
            Where Product.CategoryID = row.CategoryID _
            AndAlso Product.Discontinued = False _
            Into Sum(Product.UnitPrice * Product.UnitsInStock)
```

وبافتراض أنك متابع معي منذ البدء أصبحت تعرف طريقة الاستعلام فهنا استخدمنا Aggregate في بداية الاستعلام بما أننا نريد استخدام الدالات التجميعية للحصول على مجموع الكلفة الإجمالية للبضائع التي مازال إنتاجها مستمرا ثم حددنا في قسم Where الشرط بأننا نريد أن يتم تجميع المنتجات الموافقة للفئة المحددة row.CategoryID = Product.CategoryID وأن إنتاجها ما زال مستمرا Product.Discontinued = False ثم استخدمنا قسم Into للحصول على الإجمالي المطلوب وذلك بتمرير جداء قيمة المنتج والعدد الموجود للدالة التجميعية Sum. ولإظهار نتيجة الاستعلام على النموذج ضع صندوق نصوص على النموذج واستخدم الكود التالي لوضع القيمة فيه الذي يستخدم الأمر Format لتنسيق القيمة المعادة من الاستعلام بتنسيق عملة

```
Me.TextBox1.Text = Format(Total, "c")
```

لتنفيذ بعض التصفية على الفئات أضف تحكم صندوق نصوص وزر إلى شريط الأدوات الموجود في أعلى النموذج وفي إجراء حدث النقر على الزر أدخل الاستعلام التالي

```
Dim SelCat = From Cat In Me.NorthwindDataSet.Categories _
          Where Cat.CategoryName.ToLower Like Me.ToolStripTextBox1.Text.ToLower & "*" _
          Select Cat

Me.CategoriesBindingSource.DataSource = SelCat.AsDataView
```

حيث قمنا باستخدام Like في قسم Where لتصفية النتائج المعادة من الاستعلام تماما كما نفعل في استعلامات قواعد البيانات ثم نقوم بضبط خاصية DataSource لـ CategoriesBindingSource لكي يظهر لنا نتائج الاستعلام حيث أن الطريقة AsDataView الخاصة بالاستعلام تعيد غرض DataView داعم لـ Linq يمثل استعلام Linq to DataSet

دعنا نجري بعض الاستعلامات الأخرى وسع نافذة مشروعك قليلا وأضف DataGridView وزر أوامر جديدان عليها ثم سنستخدم كودنا السابق الذي يحصل على متغير يحمل السجل الحالي في CategoriesBindingSource للحصول على معلومات السجل الحالي الذي تم اختياره في CategoriesDataGridView وذلك في بداية إجراء معالجة الحدث Click لزر الأوامر

```
Dim row As NorthwindDataSet.CategoriesRow
row = CType(CType(Me.CategoriesBindingSource.Current, _
      DataRowView).Row, NorthwindDataSet.CategoriesRow)
```

الآن يمكننا كتابة الاستعلام التالي للحصول على قائمة بالمنتجات التي مازالت قيد الإنتاج مع السعر الإجمالي للموجود منها حاليا وإظهار النتيجة في DataGridView1 والتي من الفئة التي تم اختيارها من الشبكة الخاصة بالفئات

```
Dim Prods = From pr In NorthwindDataSet.Products _
            Where pr.CategoryID = row.CategoryID _
            And pr.Discontinued = False _
            Select pr.ProductName, pr.UnitPrice, _
            pr.UnitsInStock, Total = (pr.UnitPrice * pr.UnitsInStock)

Me.DataGridView1.DataSource = Prods.ToList
```

أضف ثلاث صناديق نصوص إلى النافذة حيث سنقوم بإنشاء استعلام جديد لحساب متوسط سعر الوحدات الموجودة ومجموع الكميات والقيمة الإجمالية وذلك من أجل نفس الفئة التي أظهرنا نتائجها في الاستعلام السابق

```
Dim ProdSums = Aggregate pr In NorthwindDataSet.Products _
               Where pr.CategoryID = row.CategoryID _
               And pr.Discontinued = False _
               Into UntiSum = Sum(pr.UnitsInStock), PriceAvg = Average(pr.UnitPrice), _
               TotalValue = Sum(pr.UnitPrice * pr.UnitsInStock)

Me.TextBox2.Text = ProdSums.PriceAvg.ToString("#,###.00")
Me.TextBox3.Text = ProdSums.UntiSum
Me.TextBox4.Text = ProdSums.TotalValue
```

لاحظ أن طريقة كتابة استعلامات Linq قريبة جدا من طريقة كتابة استعلامات select في SQL مع بعض الاختلاف في الترتيب وأن صيغة هذه الاستعلامات متشابهة مهما اختلف مزود Linq الذي نتعامل معه حيث يمكننا استخدام الأشكال المختلفة للاستعلام التي وردت في الدرس السابق المتعلق بـ Linq to Object هنا أيضا عندما يتعلق الأمر بـ Linq to Dataset وفي الدروس المستقبلية عندما نتحدث عن Linq to xml و Linq to Sql

مثال عملي على Linq To DataSet مع استخدام Lambda Expressions

الهدف من المثال

١. حفظ بيانات DataSet في ملف xml واستعادتها منه والتعامل معها بدون الحاجة لوجود قاعدة بيانات
٢. الاستعلام من الـ DataSet باستخدام Linq ومن أكثر من جدول وإدخال بعض الحسابات في جملة الاستعلام واستخدام عبارة Join لمنع التكرار الخاطئ للبيانات
٣. استخدام Lambda Expressions للقيام بالحسابات من أجلنا والاستفادة من ميزة رفع المتغيرات
٤. إظهار نتيجة استعلام Linq في DataGridView مباشرة

من أجل ترك الموضوع عام وبما أن مجموعة البيانات DataSet يمكن ربطها مع أي قاعدة بيانات سأقوم بالعمل على مجموعة بيانات غير مربوطة مع قاعدة بيانات حيث يمكنك تطبيق الأفكار الواردة هنا على أي قاعدة بيانات يمكن ربطها مع أي DataSet وسيكون مثالي الذي سنسير عليه هنا معتمد على خدمة مصرفية تدعى بالودائع لأجل

افتح أي إصدار من فيجول ستوديو 2008 وأنشئ مشروعا جديدا من نوع Windows Forms Application ثم من قائمة Project اختر الأمر Add New Item وأضف DataSet للمشروع وقم بتسميتها MyDataSet ثم في محرر التصميم الرسومي لمجموعة البيانات انقر بزر الفأرة اليميني واختر الأمر DataTable من قائمة Add في قائمة السياق ثم قم بإعادة تسميته ليصبح اسمه Customers ثم أضف للجدول Customers الحقول التالية مع الخصائص الموضحة بجانب كل منها

الحقل ID الخاصية AutoIncrement بالقيمة True والخاصيتان AutoIncrementSeed و AutoIncrementStep كلتاهما إلى القيمة 1 و نوع البيانات System.Int32 ثم انقر بزر الفأرة اليميني على الحقل ID واختر الأمر Set Primary Key من قائمة السياق لتحديد الحقل كمفتاح أساسي

الحقل CustomerName نوع البيانات System.String و MaxLength بالقيمة 25

الحقل CurrentAccountNumber نوع البيانات System.String و MaxLength 25

حيث أن الحقل ID هو معرف الزبون و CustomerName هو اسم الزبون والحقل CurrentAccountNumber هو رقم الحساب الجاري لدى المصرف

أضف جدول آخر لمجموعة البيانات باسم Wadaeaa وأضف له الحقول التالية مع الخصائص الموضحة بجانب كل منها

الحقل WadeaaNumber بنوع بيانات String

الحقل CustomerID بنوع بيانات Int32

الحقل InterestRate بنوع بيانات Decimal و NullValue مساوية لـ 7.5

الحقل WadeeaPeriod بنوع بيانات Int16 و BullValue مساوية لـ 3

الحقل StartDate بنوع بيانات DateTime

الحقل WadeaaAmount بنوع بيانات Int32 و NullValue بقيمة 10000

حيث WadeaaNumber هو رقم الوديعة و CustomerID هو حقل مرتبط بجدول الزبائن و InterestRate نسبة الفائدة و WadeeaPeriod فترة الوديعة بالأشهر و StartDate تاريخ فتح الوديعة و WadeaaAmount قيمة الوديعة

سنضيف الآن علاقة بين الجدولين: انقر بزر الفأرة اليمني على الجدول Customers ومن القائمة الفرعية Add اختر Relation ثم اضبط Parent Table إلى Customers و Child Table إلى Wadaeaa ثم اختر Key Columns ليضم الحقل ID فقط و Foreign Key Columns ليضم الحقل CustomerID فقط ثم اختر الخيار Both Relations And Foreign Key Constraint ثم اضغط Ok من أجل حفظ العلاقة ثم اختر الأمر Save All من القائمة File

انتقل إلى محرر النماذج الخاص بـ Form1 واجعل مساحة Form1 كبيرة كفاية لتتسع لـ 2 × DataGridView مع شريط أدوات وبعض التحكمات الأخرى التي سنضيفها لاحقا ثم انتقل لنافذة Data Sources واسحب الجدول Customers ثم ألقه على سطح Form1 فيتم إضافة DataGridView و شريط أدوات للنافذة

من نافذة Data Sources انقر إشارة + بجانب الجدول Customers لتظهر لك قائمة بالحقول الخاصة به كما نلاحظ وجود نسخة من الجدول Wadaeaa كجدول فرعي ضمن Customers وذلك بسبب العلاقة التي قمنا بإنشائها بين الجدولين الآن قم بسحب الجدول Wadaeaa الموجود كجدول فرعي لـ Customers وليس الجدول الخارجي إلى سطح Form1 ليتم إنشاء DataGridView أخرى على النافذة ثم قم بتنسيق النافذة بشكل جيد وتأكد من أن الـ DataGridView الخاصة بـ Customers في الأعلى و الأخرى في الأسفل

اختر CustomersDataGridView وانقر على السهم الصغير الذي يظهر في زاويتها اليمينية العليا واختر الأمر Edit Columns واضبط الخاصية Visible للحقل ID إلى False ثم كرر العملية بالنسبة للحقل CustomerID في WadaeaaDataGridView

في شريط الأدوات في الأعلى انقر بزر الفأرة اليميني على زر الحفظ – يمتلك أيقونة قرص – واختر الأمر Enabled لتفعيله ثم انقر عليه نقرا مزدوجا لننتقل إلى محرر الكود ثم أدخل الكود التالي في حدث النقر على زر الحفظ حيث نستخدم الوظيفة WriteXml لتخزين محتويات مجموعة البيانات في ملف xml

```
Try
    MyDataSet.WriteXml("d:\TestData.xml")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

أنشئ إجراء لمعالجة الحدث Load للنموذج وأدخل فيه الكود التالي الذي سيقوم بتحميل البيانات من ملف xml لاحظ ظهور رسالة خطأ عند تشغيل البرنامج لأول مرة وقبل حفظ البيانات حيث أن ملف البيانات لم يتم إنشاؤه بعد وهذا السبب في استخدام بلوك Try ... Catch من أجل اصطياد الخطأ وتجنب إفشال عملية بدء البرنامج وقد استخدمنا الوظيفة ReadXml لتحميل البيانات من ملف xml إلى مجموعة البيانات

```
Try
    MyDataSet.Clear()
    MyDataSet.ReadXml("d:\TestData.xml")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

شغل البرنامج وأدخل فيه بعض البيانات في كلا الـ DataGridView وتأكد من أنك قد قمت بتعبئة جميع الحقول في كلتا شبكتي البيانات وانتبه إلى أن الحقل WadeeaPeriod هو عبارة عن عدد أشهر فترة الوديعة لذا حاول الالتزام بالقيم 1 أو 3 أو 6 أو 12 كقيمة لهذا الحقل من أجل تجربة إجرائية الاحتساب لاحقا وقم بالحفظ وأغلق البرنامج ثم أعد فتحه من جديد للتأكد من أن عملية الحفظ قد تمت بشكل صحيح لاحظ عدم ظهور رسالة الخطأ التي ظهرت عند فتح البرنامج لأول مرة بعد أن أدخلنا بيانات وقمنا بحفظها عند تشغيل البرنامج للمرة الثانية

نريد الآن إظهار قيم الاحتسابات الخاصة بكل وديعة عند المرور عليها وكذلك تاريخ استحقاق هذه الوديعة

أضف أربعة حقول نصية للنافذة ورتبها أسفل شبكتي البيانات وأعطها الأسماء التالية txtEndDate لتاريخ الاستحقاق و txtInterest لقيمة الفائدة و txtRayaa لضريبة الريع و txtIdara لضريبة الإدارة المحلية ثم انتقل لمحرر الكود وأضف الاستيراد التالي في بداية ملف الكود الخاص بـ Form1 من أجل تمكيننا من استخدام الوظائف الموجودة في مجال الأسماء Math

```
Imports ma = System.Math
```

ثم أضف الإجراء التالي كإجرائية للاحتساب وهنا أرجو أن تكون قد تابعت دروسي المتعلقة بـ Lambda Expressions لأنها الأساس في إجرائية الاحتساب

```
Private Sub DisplayWaeaaCalcs(ByVal Amount As Integer, ByVal StartDate As Date, _
       ByVal EndDate As Date, ByVal Interest As Decimal)

        Dim DaysNum = DateDiff(DateInterval.Day, StartDate, EndDate)
        Dim Rayaa As Decimal

        Dim Rayya_Calc = Function(Intrst As Decimal) ma.Ceiling(Intrst * 7.5 / 100)
```

```
        Dim Idara_Calc = Function() ma.Ceiling(Rayaa * 10 / 100)
        Dim Intrst_Calc = Function() _
                            ma.Ceiling(Amount * DaysNum * Interest / 36500)

        Dim Intr = Intrst_Calc()
        Rayaa = Rayya_Calc(Intr)
        Dim Ida = Idara_Calc()

        Me.txtEndDate.Text = EndDate.ToString("dd/MM/yyyy")
        Me.txtInterest.Text = Intr.ToString("#,###.00")
        Me.txtRayaa.Text = Rayaa.ToString("#,###.00")
        Me.txtIdara.Text = Ida.ToString("#,###.00")
    End Sub
```

حيث استخدمنا في البداية الدالة DateDiff للحصول على عدد أيام الفترة التي سنقوم بالاحتساب عنها وكنا قد مررنا قيم المبلغ و تاريخ البداية وتاريخ النهاية ونسبة الفائدة كمحددات لإجرائية الاحتساب ثم عرفنا تعبير لمدا يقوم باحتساب قيمة ضريبة الريع بناء على مبلغ الفائدة الممررة له وفي تعبير لمدا والضريبة الأخرى لم نمرر لها قيمة ولكنها استخدمت متغير محلي من أجل الاحتساب و هنا أنصحك بمراجعة قسم رفع المتغيرات في مواضيعي المتعلقة بتعابير لمدا إن لم تكن قد قرأته حتى الآن وتعبير لمدا الأخير Intrst_Calc يستخدم أيضا خاصية رفع المتغيرات ولكنه هنا يستخدم المحددات الممررة للإجراء كمتغيرات مرفوعة ثم نقوم باستخدام هذه التعابير للاحتساب ثم نظهر القيم في صناديق النصوص المناسبة

من أجل أن نقوم باحتساب القيم الموافقة لكل وديعة عند المرور عليها سنقوم بعمل إجراء معالجة للحدث CellEnter لكلا شبكتي البيانات بإجراء واحد – أدخل الكود التالي كإجراء لمعالجة الحدث CellEnter لكلا الشبكتين لاحظ ما بعد عبارة Handles في بداية تعريف جسم الإجراء وأيضا أنني لم أقم بتمرير أية محددات لإجراء معالجة الحدث حيث يمكنني فعل ذلك بما أنني متأكد من أنني لن أحتاج لاستخدامها

```
Private Sub WadaeaaDataGridView_CellEnter() _
                        Handles WadaeaaDataGridView.CellEnter, CustomersDataGridView.CellEnter

        Try
            Dim EdDat = From a In MyDataSet.Wadaeaa _
                Where a.CustomerID = Me.CustomersDataGridView.CurrentRow.Cells(0).Value _
                And a.WadeaaNumber = Me.WadaeaaDataGridView.CurrentRow.Cells(0).Value _
                Select a.WadeaaNumber, a.WadeaaAmount, a.StartDate, _
                EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate), _
                a.InterestRate

            If EdDat.Count > 0 Then
                DisplayWaeaaCalcs(EdDat.First.WadeaaAmount, EdDat.First.StartDate, _
                                  EdDat.First.EndDate, EdDat.First.InterestRate)
            End If
        Catch ex As Exception
            Me.txtEndDate.Text = String.Empty
            Me.txtInterest.Text = String.Empty
            Me.txtRayaa.Text = String.Empty
            Me.txtIdara.Text = String.Empty
        End Try
    End Sub
```

في البداية قمنا بإنشاء استعلام Linq للحصول على القيم الخاصة بالوديعة التي نقف عليها حيث أن المتغير a هو كيان من الجدول Wadaeaa ثم في قسم Where ضبطنا الشرط بحيث يجلب الاستعلام فقط الودائع الخاصة بزبون معين عن طريق التأكد من أن قيمة الحقل CustomerID مساوية لقيمة ID الخاصة بالزبون من خلال قراءة قيمة الخلية المناسبة في السطر الحالي وتتمة للشرط وبنفس الطريقة قمنا بضبط الشرط كي يجلب الوديعة ذات الرقم المراد ثم يأتي قسم Select لنحدد فيه قائمة الحقول التي نريد الحصول عليها لاحظ وجود الحقل المحسوب EndDate الذي يتم حساب قيمته من الحقول المعادة من الاستعلام باستخدام الوظيفة DateAdd التي تضيف فترة زمنية معينة حسب المحددات الممررة لها إلى تاريخ ممرر لها وتعيد قيمة التاريخ الجديد وتعاد قيمته مع قائمة الحقول التي يعيدها الاستعلام وبعد الاستعلام نتأكد من أنه قد جلب نتائج فعلا بالتحقق من قيمة الخاصية Count ثم نستدعي الوظيفة DisplayWaeaaCalcs للقيام بالحسابات وإظهار النتائج

شغل البرنامج ولاحظ ظهور قيم الاحتسابات في مربعات النصوص عند التنقل في كلا شبكتي البيانات إذا كانت لديك بيانات قمت بحفظها كما طلبت منك سابقا

من أجل إظهار الودائع التي تبدأ بتاريخ معين وإظهارها أضف نموذج آخر للمشروع باسم Form2 ثم أضف DataGridView له واضبط الخاصية Dock للقيمة Fill لجعل شبكة البيانات تملأ كامل مساحة النموذج ثم نسق النموذج بحيث يكون كبيرا كفاية لعرض البيانات الناتجة عن الاستعلام ثم أضف زرا للنموذج From1 واجعل إجراء معالجة حدث النقر عليه يماثل الكود التالي

```
Private Sub Button1_Click() Handles Button1.Click
    Dim d As Date = CDate(InputBox("Enter Date"))
    Dim Dawa = From a In MyDataSet.Wadaeaa _
               Join B In MyDataSet.Customers On _
               a.CustomerID Equals B.ID _
               Where a.StartDate = d _
               Select B.CustomerName, B.CurrentAccountNumber, _
               a.WadeaaNumber, a.WadeaaAmount, a.WadeeaPeriod, a.StartDate, _
               EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, _
                                 a.StartDate)

    Dim c As New Form2
    c.DataGridView1.DataSource = Dawa.ToList
    c.DataGridView1.Update()
    c.ShowDialog()
 End Sub
```

لاحظ أنني استخدمت ميزة جديدة في فيجول ستوديو تمكنني من حذف محددات إجراء معالجة حدث ما إن كنت على يقين أنني لن أحتاج لاستخدامها وفي جملة الاستعلام تلاحظ أنني استخدمت Join للربط بين الجداول عند عملية الاستعلام كي نتجنب مشكلة ظهور بيانات مكررة من أحد الجداول من أجل جميع سطور الجدول الآخر حيث استخدمنا نفس أسلوب العلاقة التي قمنا بإنشائها في البداية بين الجدولين من حيث ربط الحقل CustomerID في الجدول Wadaeaa بالحقل ID في الجدول Customer و استخدمنا في قسم Where شرط لتصفية نتائج الاستعلام بحيث نحصل على الودائع التي تبدأ بتاريخ معين ثم نستخدم عبارة Select لتحديد الحقول التي نريد إظهارها كنتائج للاستعلام

ومن أجل إظهار النتائج في Form2 قمنا بإنشاء متغير من نوع تلك النافذة ثم ضبطنا قيمة DataSource لشبكة البيانات الموجودة على ذلك النموذج إلى النتيجة المعادة من الاستعلام مستخدمين الطريقة ToList لتحويل النتائج إلى شكل يمكن إظهاره في شبكة البيانات ومن أجل الحصول على الودائع التي تنتهي بتاريخ معين يمكننا استخدام نفس الكود السابق بعد تعديل بسيط في قسم Where بحيث يصبح الكود كما يلي

```
Private Sub Button2_Click() Handles Button2.Click
    Dim d As Date = CDate(InputBox("Enter Date"))
    Dim Dawa = From a In MyDataSet.Wadaeaa _
            Join B In MyDataSet.Customers On _
            a.CustomerID Equals B.ID _
            Where DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate) = d _
               Select B.CustomerName, B.CurrentAccountNumber, a.WadeaaNumber, _
               a.WadeaaAmount, a.WadeeaPeriod, a.StartDate, _
              EndDate = DateAdd(DateInterval.Month, a.WadeeaPeriod, a.StartDate)

    Dim c As New Form2
    c.DataGridView1.DataSource = Dawa.ToList
    c.DataGridView1.Update()
    c.ShowDialog()
End Sub
```

لاحظ الاختلاف في قسم Where بين الإجراءين الأخيرين. هل يمكنك شرح عمل الإجراء الثاني بنفسك ؟؟؟؟

مقدمة في Linq to XML

Linq to Xml هي واجهة برمجة xml في الذاكرة تدعم Linq تمكنك من العمل مع بيانات xml المختلفة من داخل لغة برمجة الـ .net Framework وهي مشابهة لـ Document Object Model واختصارا DOM التي تضع الـ xml في الذاكرة حيث يمكنك الاستعلام من الوثيقة أو التعديل عليها ثم يمكنك حفظها أو إرسالها بعد التعديل ولكن تختلف Linq to xml عن DOM في أنها تزود نموذج غرضي Object Model أخف وأسهل عند العمل عليه وهي تستفيد من تطويرات اللغة في الـ2008

وتكمن الميزة الأهم التي تقدمها Linq to Xml هي التكامل مع Linq الذي يمكنك من كتابة استعلامات من وثيقة xml في الذاكرة للحصول على مجموعة من العناصر والصفات التي تمتد لتشمل xPath و Xquery وتقدم لك ميزات إضافية مثل اكتشاف الأخطاء في وقت الترجمة ودعم أفضل للمدقق Debugger إضافة إلى ترميز أقوى وإمكانية استخدام نتائج الاستعلامات كوسائط لبانيات XElement أو XAttribute توفر طريقة سهلة لإنشاء أشجار xml وهي تدعى Functional Construction التي تمكن المطورين بسهولة من تحويل أشجار xml بسهولة من شكل إلى آخر.

وتمكنك إمكانيات Linq في Linq to xml من كتابة استعلامات من xml فقد يكون لديك ملف xml يمثل طلب مشتريات كالتالي

PurchaseOrder.xml

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

فباستخدام Linq to xml يمكنك تشغيل استعلام للحصول على القيمة المقابلة للصفة PartNumber من أجل كل عنصر في طلب المشتريات

```
Dim purchaseOrder As XElement = XElement.Load("PurchaseOrder.xml", LoadOptions.SetBaseUri
Or LoadOptions.SetLineInfo)

Dim partNos = _
    From item In purchaseOrder...<Item> _
    Select item.@PartNumber
```

وقد تريد الحصول على قائمة مرتبة باستخدام PartNumber بالعناصر التي تحمل القيمة أكثر من 100 وللحصول على هذه المعلومات يمكننا كتابة الاستعلام

```
Dim partNos = _
    From item In purchaseOrder...<Item> _
    Where (item.<Quantity>.Value * _
           item.<USPrice>.Value) > 100 _
    Order By item.<PartNumber>.Value _
    Select item
```

وباستخدام Linq to xml يمكنك عمل العديد من الأشياء كتحميل ملف من القرص أو تخزين ملف إلى القرص أو إنشاء بيانات xml من الصفر أو الاستعلام باستخدام Xpath أو حتى التعامل مع أشجار xml من حيث الإضافة والحذف والتعديل والتأكد من صحة أشجار xml باستخدام XSD أو استخدام مجموعة مما ورد هنا لتحويل أشجار xml من شكل إلى آخر

وهناك طريقتان لإنشاء أشجار xml في Visual Basic إما بتعريف xml مباشرة في الكود أو باستخدام Linq APIs لإنشاء الشجرة وكلتا الطريقتين تمكنان الكود من عكس بنية xml شجرية كاملة فالكود التالي مثلا ينشئ عنصر xml

```
Dim contact1 As XElement = _
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

ويقدم فيجول بايزيك عدة خصائص للتنقل عبر بنية xml والتي تمكنك من الوصول إلى عناصر وصفات xml عن طريق تحديد اسم عنصر xml الابن أو يمكنك استدعاء طرائق Linq لتحديد العناصر الأبناء لعنصر xml فالكود التالي مثلا يستخدم خصائص xml للإشارة إلى الصفات والعناصر الأبناء لعنصر xml مستخدما استعلام Linq للحصول على العناصر الأبناء وإخراجهم كعنصر xml

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module Sample1

    Sub SampleTransform()

        ' Create test by using a global XML namespace prefix.

        Dim contact = _
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        Dim phoneTypes = _
          <phoneTypes>
              <%= From phone In contact.<ns:phone> _
                  Select <type><%= phone.@ns:type %></type> _
              %>
          </phoneTypes>

        Console.WriteLine(phoneTypes)
    End Sub

End Module
```

ويمكنك Visual Basic من تحديد اسم مستعار Alias لمجال أسماء Xml باستخدام عبارة Imports كما في الكود التالي الذي يرينا كيف يمكننا استخدام العبارة Imports لاستيراد مجال أسماء XML

```
Imports <xmlns:ns="http://someNamespace">
```

حيث يمكنك استخدام هذا الاسم المستعار للوصول إلى خصائص xml ولتحديد محارف xml من أجل وثيقة وعناصر xml ويمكننا الحصول على غرض XNamespace من أجل أي بادئة مجال أسماء باستخدام المعامل GetXmlNamespace كما في المثال

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module GetXmlNamespaceSample

    Sub RunSample()

        ' Create test by using a global XML namespace prefix.

        Dim contact = _
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        ShowName(contact.<ns:phone>(0))
    End Sub

    Sub ShowName(ByVal phone As XElement)
        Dim qualifiedName = GetXmlNamespace(ns) + "contact"
        Dim contact = phone.Ancestors(qualifiedName)(0)
        Console.WriteLine("Name: " & contact.<ns:name>.Value)
    End Sub

End Module
```

ويرينا المثال التالي كيفية إنشاء XElemnt باستخدام مجال الأسماء العام ns

```
Dim contact1 As XElement = _
    <ns:contact>
        <ns:name>Patrick Hines</ns:name>
        <ns:phone type="home">206-555-0144</ns:phone>
        <ns:phone type="work">425-555-0145</ns:phone>
    </ns:contact>

Console.WriteLine(contact1)
```

ويقوم مترجم فيجول بايزيك بترجمة محارف xml التي تحتوي الأسماء المستعارة لمجالات أسماء Xml إلى الكود المكافئ الذي يستخدم تدوين Xml المستخدم في تلك المجالات وباستخدام الصفة xmlns عند الترجمة والكود السابق يولد نفس الكود التنفيذي الذي يولده الكود التالي

```
Dim contact2 As XElement = _
    <ns1:contact xmlns:ns1="http://someNamespace">
        <ns1:name>Patrick Hines</ns1:name>
        <ns1:phone type="home">206-555-0144</ns1:phone>
        <ns1:phone type="work">425-555-0145</ns1:phone>
    </ns1:contact>

Console.WriteLine(contact2)
```

يمكن استخدام مجالات أسماء Xml العامة مع خصائص Xml كما في المثال التالي

```
Console.WriteLine("Contact name is: " & contact1.<ns:name>.Value)
```

بعض استخدامات Linq TO XML

يمكننا استخدام Linq لإنشاء وثائق xml في فيجول بايزك انظر الكود التالي الذي يقوم بإنشاء وثيقة xml تحتوي معلومات عن العمليات الجارية في النظام

```
Dim xmlProc = <MyProcesses>
                  <%= From proc In System.Diagnostics.Process.GetProcesses() _
                      Select <process id=<%= proc.Id %>>
                                 <name><%= proc.ProcessName %></name>
                                 <threads><%= proc.Threads.Count %></threads>
                             </process> %>
              </MyProcesses>

My.Computer.FileSystem.WriteAllText("d:\temp\processes.xml", xmlProc.ToString, False)
Process.Start("d:\temp\processes.xml")
```

حيث أنشأنا العقدة MyProcesses وأدخلنا فيها بداية الاستعلام فيها ثم في قسم select أنشأنا العقد الفرعية وأدخلنا بقية الاستعلام ليقوم بضبط قيم تلك العقد. كما يمكننا فيجول بايزيك من الحصول على معلومات عن العقد في وثيقة xml بسهولة حيث نستخدم الكود التالي لإظهار معلومات من الوثيقة التي قمنا بإنشائها سابقا في ListBox

```
Dim xmlprocs1 = xmlProc...<process>
For Each a In xmlprocs1
    Me.ListBox1.Items.Add(a.<name>.Value & " " & a.@id)
Next
```

أو يمكننا الحصول على نفس النتيجة السابقة باستخدام استعلامات Linq للاستعلام من وثيقة xml كما في الكود

```
Dim xmlprocs1 = From pr In xmlProc...<process> _
                Select pr.<name>.Value, pr.@id

For Each a In xmlprocs1
    Me.ListBox1.Items.Add(a.name. & " " & a.id)
Next
```

وقد نريد إنشاء وثيقة xml كنتيجة لاستعلام من وثيقة موجودة سابقا على القرص ويجب الانتباه إلى أن الاستعلام من وثائق xml حساس لحالة الأحرف

```
Dim myCusts = XDocument.Load("c:\MyCustomers.xml")

Dim ukCustomers = <ukCustomers>
                      <%= From cust In myCusts...<Customer> _
                          Where cust.<Country>.Value = "UK" _
                          Select cust %>
                  </ukCustomers>
```

أو يمكننا كتابة استعلام مباشر من وثيقة xml كما يلي

```
Dim xmlPlant = XDocument.Load(CurDir() & "\plants.xml")

Dim qa = From p In xmlPlant...<PLANT> _
         Select name = p.<COMMON>.Value _
         Order By name
```

وقد نريد كتابة استعلام من ذلك الملف لنحصل على النباتات التي تكلف أكثر من 5

```
Dim qb = From p In xmlPlant...<PLANT> _
         Where CInt(p.<PRICE>.Value) > 5 _
         Select name = p.<COMMON>.Value
```

وقد نريد إنشاء وثيقة إكسل من استعلام Linq To xml لذا افتح Excel وادخل في الثلاث خلايا الأولى من السطر الأول الكلمات التالية بالتسلسل Name و Phone و Country ثم اجعل الخط سميكا ثم أدخل تحتها سطرا من البيانات التجريبية كي نستخدمه في تحديد القسم الذي سنضع فيه بياناتنا لاحقا ثم احفظ الملف بصيغة XML SpreadSheet ثم قم بفتح الملف الذي أنشأته للتو بواسطة Notepad وانسخ جميع محتوياته ثم عد إلى محرر الكود في بيئة التطوير واكتب = Dim Sheet ثم ألصق بعدها محتويات الحافظة فيصبح لديك شيئا شبيها بالتالي

```
Dim Sheet = <?xml version="1.0"?>
            <?mso-application progid="Excel.Sheet"?>
            <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
                xmlns:o="urn:schemas-microsoft-com:office:office"
                xmlns:x="urn:schemas-microsoft-com:office:excel"
                xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
                xmlns:html="http://www.w3.org/TR/REC-html40">
                <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
                    <Author>SamerSelo</Author>
                    <LastAuthor>SamerSelo</LastAuthor>
                    <Created>2008-09-19T20:31:43Z</Created>
                    <Version>12.00</Version>
                </DocumentProperties>
                <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
                    <WindowHeight>7140</WindowHeight>
                    <WindowWidth>15255</WindowWidth>
                    <WindowTopX>120</WindowTopX>
                    <WindowTopY>150</WindowTopY>
                    <ProtectStructure>False</ProtectStructure>
                    <ProtectWindows>False</ProtectWindows>
                </ExcelWorkbook>
                <Styles>
                    <Style ss:ID="Default" ss:Name="Normal">
                        <Alignment ss:Vertical="Bottom"/>
                        <Borders/>
                        <Font ss:FontName="Arial" x:CharSet="178" x:Family="Swiss" ss:Size="11"
                            ss:Color="#000000"/>
                        <Interior/>
                        <NumberFormat/>
                        <Protection/>
                    </Style>
                </Styles>
                <Worksheet ss:Name="ورقة1" ss:RightToLeft="1">
                    <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="2" x:FullColumns="1"
                        x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
                        <Row>
                            <Cell><Data ss:Type="String">Name</Data></Cell>
                        <Cell><Data ss:Type="String">Phone</Data></Cell>
                             <Cell><Data ss:Type="String">Country</Data></Cell>
                        </Row>
                         <Row>
                         <Cell><Data ss:Type="String">Test</Data></Cell>
                              <Cell><Data ss:Type="Number">123456</Data></Cell>
                         <Cell><Data ss:Type="String">Syr</Data></Cell>
                         </Row>
                    </Table>
                    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
                        <PageSetup>
                            <Header x:Margin="0.3"/>
                            <Footer x:Margin="0.3"/>
                            <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
                        </PageSetup>
                        <Selected/>
                        <DisplayRightToLeft/>
                        <Panes>
                            <Pane>
                                <Number>3</Number>
                                <ActiveRow>1</ActiveRow>
                                <ActiveCol>2</ActiveCol>
                             </Pane>
                        </Panes>
                        <ProtectObjects>False</ProtectObjects>
                        <ProtectScenarios>False</ProtectScenarios>
                    </WorksheetOptions>
                </Worksheet>
                <Worksheet ss:Name="ورقة2" ss:RightToLeft="1">
                    <Table ss:ExpandedColumnCount="1" ss:ExpandedRowCount="1" x:FullColumns="1"
                        x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
                    </Table>
                    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
```

```
                <PageSetup>
                    <Header x:Margin="0.3"/>
                    <Footer x:Margin="0.3"/>
                    <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
                </PageSetup>
                <DisplayRightToLeft/>
                <ProtectObjects>False</ProtectObjects>
                <ProtectScenarios>False</ProtectScenarios>
            </WorksheetOptions>
        </Worksheet>
        <Worksheet ss:Name="ورقة3" ss:RightToLeft="1">
            <Table ss:ExpandedColumnCount="1" ss:ExpandedRowCount="1" x:FullColumns="1"
                x:FullRows="1" ss:DefaultColumnWidth="54" ss:DefaultRowHeight="14.25">
            </Table>
            <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
                <PageSetup>
                    <Header x:Margin="0.3"/>
                    <Footer x:Margin="0.3"/>
                    <PageMargins x:Bottom="0.75" x:Left="0.7" x:Right="0.7" x:Top="0.75"/>
                </PageSetup>
                <DisplayRightToLeft/>
                <ProtectObjects>False</ProtectObjects>
                <ProtectScenarios>False</ProtectScenarios>
            </WorksheetOptions>
        </Worksheet>
    </Workbook>
```

نلاحظ في بداية وثيقة xml التي ألصقناها للتو وجود بعض مجالات الأسماء الخاصة بـ xml في بدايتها وسنحتاج لاستيرادها في بداية كودنا لذا في قسم الاستيرادات في بداية الملف أدخل الاستيرادات التالية حتى تساعدنا في معرفة أسماء العناصر عند كتابة الاستعلام

```
Imports <xmlns="urn:schemas-microsoft-com:office:spreadsheet">
Imports <xmlns:o="urn:schemas-microsoft-com:office:office">
Imports <xmlns:x="urn:schemas-microsoft-com:office:excel">
Imports <xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
```

من محرر الكود ابحث عن الجزء الذي يمثل البيانات التجريبية ثم قم بقصه وفي حالة مثالي سيكون

```
<Row>
    <Cell><Data ss:Type="String">Test</Data></Cell>
    <Cell><Data ss:Type="Number">123456</Data></Cell>
    <Cell><Data ss:Type="String">Syr</Data></Cell>
</Row>
```

أدخل الآن الاستعلام التالي قبل المتغير sheet الذي عرفناه سابقا حيث سنستخدم فيه قطعة النص المقصوصة لتشكيل شكل ناتج الاستعلام

```
Dim Customers = From Customer In db.Customers _
                Order By Customer.CompanyName _
                Select <Row>
                           <Cell><Data ss:Type="String"><%= Customer.CompanyName %></Data></Cell>
                           <Cell><Data ss:Type=" String"><%= Customer.Phone %></Data></Cell>
                           <Cell><Data ss:Type="String"><%= Customer.Country %></Data></Cell>
                       </Row>
```

ثم انتقل للمكان الذي قصصنا منه قطعة xml سابقا وقم بإدخال السطر التالي في نفس المكان

```
<%= Customers %>
```

انتقل إلى الأعلى قليلا وعدل السطر

```
<Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="2" x:FullColumns="1"
```

إلى

```
<Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount=<%= Customers.Count + 1 %> x:FullColumns="1"
```

وكل ما تبقى هو الحفظ وعرض الملف وهذا يتم بالكود التالي

```
Sheet.Save("d:\temp\customers11.xml")
Process.Start("d:\temp\customers11.xml")
```

تعرف على Linq to SQL و O/R Designer

سنقوم هنا بإنشاء برنامج بسيط يعرض لنا كيفية استخدام O/R Designer لإنشاء Entity Classes للتعامل المباشر مع قاعدة بيانات موجودة في SQL Server وسأعتمد حاليا على قاعدة بيانات NorthWind الشهيرة التي يمكنك الحصول عليها بسهولة من موقع مايكروسوفت وتثبيتها لديك كما يتوجب عليك إنشاء اتصال لتلك القاعدة من داخل بيئة التطوير في نافذة Server Explorer حتى يسهل علينا العمل على كل حال موضوع تركيب قاعدة البيانات وإنشاء الاتصال خارج عن مجال دورتنا وأفترض أنه لديك بعض الأساسيات التي تساعدك في التعامل مع هكذا أمور ويمكنك مراجعة مكتبة MSDN وبعض فيديوهات مايكروسوفت التعليمية إن احتجت لمساعدة في هذه الأمور

من Solution Explorer انقر بزر الفأرة اليميني على مشروعك ومن القائمة اختر Add New ومن صندوق الحوار اختر Linq To SQL Classes ثم قم بتسمية الفئة الجديدة NorthWind.dbml ثم اضغط Add فيفتح لك واجهة O/R Designer الفارغة وإن كانت مغلقة يمكنك النقر المزدوج على NorthWind.dbml من مستكشف الحل لفتحها

وسع الاتصال الخاص بقاعدة البيانات Northwind من Server Explorer واسحب الجدول Customers إلى واجهة O/R Designer ثم قم بالحفظ وبعدها انتقل إلى نافذة Data Sources اضغط Add New Data Source فيفتح لك المعالج المألوف وفي حالتنا هذه سنختار Object بما أنه المناسب لعملنا هنا ثم نضغط Next ثم وسع العقد في نافذة المعالج واختر Customer ثم اضغط Next ثم Finish

عد إلى محرر النماذج وقم بسحب الجدول Customer من واجهة DataSources إلى سطح النافذة فيتم إنشاء شريط أدوات و DataGridView من أجلك افتح السهم الصغير أعلى يمين DataGridView واضبط الخيار Dock in parent Container لجعل شبكة البيانات تملأ كافة المساحة الفارغة في النموذج وكما تلاحظ هنا أن بيئة التطوير تسهل علينا الكثير من الأمور هنا من إنشاء للتحكمات والفئات والربط بينها مما يوفر علينا الكثير من العمل

من أجل إظهار البيانات على النموذج سنحتاج لكتابة بعض الكود لذا انتقل لمحرر الكود الخاص بالنموذج وأنشئ معالج للحدث Load للنموذج وقبل بداية تعريف الحدث Load الخاص بالنموذج أدخل المتغير التالي بحيث يكون عاما على مستوى النموذج حيث أن المتغير Db هنا هو كيان من NorthWindDataContext والتي تشكل اتصالنا الفعلي مع قاعدة البيانات بما أنها نقطة الدخول الرئيسية بالنسبة لـ Linq To SQL

```
Private Db As New NorthWindDataContext
```

في الحدث Load للنموذج سنضع استعلام Linq يزودنا بالبيانات التي سيتم إظهارها

```
Dim AllCustomers = From cust In Db.Customers _
                   Order By cust.CustomerID _
                   Select cust
```

هذا استعلام Linq عادي كأي استعلام Linq قمنا باستخدامه منذ بداية الدورة حتى الآن كل ما علينا لإظهار البيانات هو ضبط قيمة الخاصية DataSource لـ CustomerBindingSourec إلى استعلامنا AllCustomers أدخل السطر التالي مباشرة بعد الاستعلام ثم شغل البرنامج وتأكد من ظهور البيانات

```
Me.CustomerBindingSource.DataSource = AllCustomers
```

من أجل حفظ التعديلات التي ربما سنقوم بها إلى قاعدة البيانات عد إلى محرر النماذج واجعل زر الحفظ الموجود على شريط الأدوات Enabled ثم انقر عليه نقرا مزدوجا لإنشاء معالج لحدث النقر عليه والانتقال لمحرر الكود وأدخل الكود التالي الذي سيقوم بحفظ البيانات من أجلنا

```
Me.Validate()
Me.CustomerBindingSource.EndEdit()

Try
    Db.SubmitChanges()
    MsgBox("Changes Saved")
Catch ex As Exception
    MsgBox(ex.Message)
```

```
End Try
```

حيث استخدمنا Me.Validate أولا لجعل جميع التحكمات على النموذج أن تتحقق من قيمها ثم استدعينا الطريقة EndEdit العائدة لـ CustomerBindingSource من أجل التأكد من أن جميع عمليات التحرير على البيانات قد تم إنهاؤها ثم استخدمنا الطريقة SubmitChanges العائدة لـ DataContext التي ستقوم بحفظ البيانات فعليا إلى قاعدة البيانات واستخدمنا بلوك Try ... Catch من أجل التقاط أي خطأ ربما نصادفه أثناء عملية الحفظ ومع أننا استخدمنا الطريقة SubmitChanges هنا بدون محددات إلا أنه يمكن استخدامها بتمرير محدد وحيد من نوع التعداد ConflictMode الذي يمتلك إحدى قيمتين FailOnFirstConflict التي توقف عملية تحديث البيانات إلى قاعدة البيانات عند حصول أول تضارب وهي القيمة الافتراضية و ContinueOnConflict وهي تتابع عملية الحفظ حتى لو حدثت تضاربات وتقوم بتجميع هذه التضاربات وتعيدها بعد انتهاء عملية تحديث البيانات إلى قاعدة البيانات

يمكنك عند هذه النقطة تشغيل البرنامج والتأكد من أنك تستطيع إضافة وتعديل وحذف البيانات بدون أية مشاكل

نريد الآن إضافة بعض التصفية على البيانات في النموذج ولهذا الغرض قم بإضافة صندوق نصوص وزر أوامر إلى شريط الأدوات في أعلى النموذج وانقر نقرا مزدوجا على الزر حتى ننتقل لمحرر الكود وفي إجراء معالجة حدث النقر عليه أدخل الآن الاستعلام التالي الذي أتبعناه بكود تحديث الخاصية DataSource لـ CustomerBindingSource إلى الاستعلام الجديد لإظهار النتائج

```
Dim CustNameQuery = From cust In Db.Customers _
                    Where cust.ContactName Like _
                    Me.ToolStripTextBox1.Text & "*" _
                    Order By cust.CustomerID _
                    Select cust

Me.CustomerBindingSource.DataSource = CustNameQuery
```

ملاحظة: أنا لا أقوم بشرح الاستعلامات هنا بافتراض أنك متابع ممتاز معي منذ البداية وأصبحت متآلفا مع صيغة وطريقة كتابة هذه الاستعلامات كما يمكنك كتابة أية استعلامات تريدها هنا وبأي شكل هنا تماما كما فعلنا في الدروس السابقة

Linq To Sql Master/Detail

افتح مشروع فيجول بايزيك جديد وقم بإضافة Linq To SQL Classes إليه تماما كما فعلنا في الدرس السابق وسمها NorthWind.dbml ثم من Server Explorer وسع عقدة اتصال قاعدة البيانات NorthWind وقم باختيار الجدولين Orders و Customers وقم بسحبهما معا إلى نافذة O/R Designer الأمر الذي ينشئ فئتين من أجلنا الأولى تدعى Customer و الثانية Order ونرى أن بيئة التطوير قامت بضبط العلاقة بينهما تلقائيا كما نلاحظ أن جميع خصائص كلتا الفئتين تماثل تماما الحقول الموجودة في الجدولين الموجودين في قاعدة البيانات قم بحفظ المشروع الآن قبل المتابعة

انتقل إلى نافذة Data Sources واختر Add New Data Source واختر من الصفحة الأولى في المعالج أن النوع الذي نريد الاتصال به هو Object ثم اضغط Next للانتقال للصفحة التالية من المعالج ثم وسع العقد وقم باختيار الجدول Customer فقط بدون اختيار الجدول Order ثم اضغط Next ثم Finish وستلاحظ في نافذة Data Sources أنه قد تم إدراج الفئة Customer وتظهر الفئة Orders ككائن فرعي منها ولهذا قمنا باختيار Customer فقط في المعالج

اسحب Customer إلى سطح النافذة ليتم إنشاء DataGridView على النافذة وشريط أدوات BindingNavigator في أعلى النافذة ثم قم بسحب Orders إلى سطح النافذة لإنشاء DataGridView أخرى أسفل الأولى

انتقل إلى محرر الكود الخاص بالنموذج وقم بإنشاء إجراء لمعالجة حدث Load للنموذج وقبل تعريف الإجراء مباشرة قم بإدخال المتغير العام التالي على مستوى النموذج ليكون كيانا من NorthWindDataContext والتي تشكل اتصالنا الفعلي مع قاعدة البيانات بما أنها نقطة الدخول الرئيسية بالنسبة لـ Linq To SQL

```
Private Db As New NorthWindDataContext
```

ولجعل البيانات تظهر في كلتا الشبكتين أدخل الكود التالي في إجراء الحدث Load للنموذج وقم بتجربة البرنامج

```
Me.CustomerBindingSource.DataSource = Db.Customers
```

وعند تجربة البرنامج ستجد أن بيانات Orders قد تم جلبها وهذا تقوم به من أجلنا Linq to Sql في الخلفية بسبب العلاقة الموجودة بين الجدولين عندما أنشأنا Entity Classes في بداية العمل بواسطة O/R Designer والذي يحدث فعليا هو أنك عندما تختار سجلا من Customers يتولد تلقائيا استعلام يجلب بيانات Orders المرتبطة بهذا السجل بموجب العلاقة بين الجدولين ويظهرها في شبكة البيانات الثانية

لكي نستطيع حفظ أية تعديلات أو إضافات نقوم بها على قاعدة البيانات عد إلى محرر النماذج واجعل زر الحفظ الموجود على الشريط في أعلى النموذج Enabled وانقر عليه نقرا مزدوجا من أجل إنشاء إجراء معالجة لحدث النقر عليه والانتقال لمحرر الكود وسيكون كود الحفظ شبيها بالكود الذي استخدمناه في الدرس السابق

```
Me.Validate()
Me.OrdersBindingSource.EndEdit()
Me.CustomerBindingSource.EndEdit()

Try
    Me.Db.SubmitChanges()
    MsgBox("Changes Saved.")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

جرب البرنامج وتأكد من أن جميع عمليات الحذف والتعديل والإضافة تعمل

انتقل إلى محرر النماذج وأضف ComboBox لشريط الأدوات في أعلى النموذج الذي سنقوم بملئه بأسماء الدول المتوفرة حتى نستطيع اختيار الزبائن الموجودين في دولة معينة انتقل الآن لمحرر الكود واستبدل كامل محتويات الحدث Load للنموذج بالكود التالي الذي سيقوم

بملئ صندوق القائمة بأسماء الدول لاحظ استخدام distinct في عبارة الاستعلام كي لا يجلب لنا نتائج مكررة عندما تكون هناك نتائج متشابهة معادة من قاعدة البيانات

```
Dim CusCountry = From co In Db.Customers _
                 Where co.Country <> String.Empty _
                 Order By co.Country _
                 Select co.Country Distinct

Me.ToolStripComboBox1.Items.Clear()

For Each co In CusCountry
    Me.ToolStripComboBox1.Items.Add(co)
Next
```

أنشئ إجراء معالجة لحدث SelectedIndexChanged لـ ToolStripComboBox1 وأدخل فيه كود الاستعلام التالي لكي يتم إظهار الزبائن الموجودة في دولة معينة عند اختيارها من صندوق القائمة المركبة

```
Dim CustQuery = From co In Db.Customers _
                Where co.Country = Me.ToolStripComboBox1.SelectedItem.ToString _
                Select co

Me.CustomerBindingSource.DataSource = CustQuery
```

وللحصول على مجموع أجور الشحن لطلبات الزبون المحدد أضف صندوق نصوص أسفل النموذج ثم أنشئ إجراء لمعالجة حدث CurrentChanged الخاص بـ CustomerBindingSource وأدخل فيه الكود التالي

```
Dim ro = CType(Me.CustomerBindingSource.Current, Customer)

Dim FriSum = Aggregate ord In Db.Orders _
             Where ord.CustomerID = ro.CustomerID _
             Into Sum(ord.Freight)

Me.TextBox1.Text = FriSum.ToString
```

حيث حصلنا أولا على معلومات السجل الحالي في CustomerBindingSource وهنا اضطررنا لاستخدام CType لتحويل ناتج الخاصية Current إلى النوع المطلوب بما أنها تعيد قيمة من النوع Object وبما أننا لم نحدد نوع المتغير ro عند التصريح عنه فستقوم بذلك نيابة عنا ميزة الاستدلال المحلي على النوع ثم استخدمنا استعلام aggregate لحساب مجموع أجور شحن الطلبات الخاصة بالزبون الحالي

أضف زرا وصندوق نصوص إلى النموذج حتى نستطيع إظهار طلبات الزبون المحدد بعد تاريخ معين ولكي لانفقد الارتباط بين شبكتي البيانات انتقل إلى إجراء معالجه الحدث CurrentChanged الخاص بـ CustomerBindingSource وأدخل فيه الكود التالي قبل الكود الموجود فيه كي نعيد ضبط القيم الصحيحة لـ OrdersBindingSource بما أننا سنغيرها لاحقا عندما سننفذ استعلامنا

```
Me.OrdersBindingSource.DataSource = CustomerBindingSource
Me.OrdersBindingSource.DataMember = "Orders"
Me.TextBox2.Text = String.Empty
```

أنشئ إجراء لمعالجة حدث النقر على الزر الذي أضفناه للنموذج وأدخل فيه الكود التالي

```
Dim ro = CType(Me.CustomerBindingSource.Current, Customer)

If IsDate(Me.TextBox2.Text) Then
    Dim SpOrd = From ord In Db.Orders _
                Where ord.CustomerID = ro.CustomerID _
                And ord.OrderDate >= CDate(Me.TextBox2.Text) _
                Select ord

    Me.OrdersBindingSource.DataSource = SpOrd
Else
    Me.OrdersBindingSource.DataSource = CustomerBindingSource
    Me.OrdersBindingSource.DataMember = "Orders"
```

```
    Me.TextBox2.Text = String.Empty
End If
```

حيث حصلنا في البداية على معلومات السجل الحالي بالنسبة لـ Customers في المتغير ro ثم استخدمنا عبارة If و الدالة CDate للتحقق من أن المستخدم قد قام بإدخال تاريخ صحيح قبل المتابعة بالاستعلام وإن لم يكن تاريخا صحيحا نعيد ضبط قيم OrdersBindingSource إلى القيم الأصلية ونقوم بتفريغ النص الموجود في صندوق النصوص وإن كان قد ادخل تاريخا صحيحا ننشئ استعلام يقوم بجلب الطلبات التي تاريخها بعد التاريخ الموجود في صندوق النصوص والخاصة بالزبون الحالي كما هو ظاهر في قسم Where في الاستعلام ثم نقوم بضبط الخاصية DataSource لـ OrdersBindingSource إلى استعلامنا كي يتم عرض البيانات المعادة من الاستعلام في شبكة البيانات الثانية

مثال سريع عن كيفية إنشاء فئات Linq To SQL يدويا

رأينا في الدروس السابقة كيف أن الـ O/R Designer يقوم بإنشاء فئات Linq To SQL أو كما تدعى أيضا Entity Classes بسهولة من أجلنا ومع ذلك يمكننا القيام بذلك يدويا وسأقوم في هذا الدرس باستعراض سريع لكيفية عمل ذلك من أجل العلم بالشئ

أنشئ مشروعا جديدا من النوع Console Application وسمه LinqConsoleApp

- من قائمة Project اختر Add Reference ومن صفحة net. اختر System.Data.Linq ثم اضغط Ok
- في بداية الملف في أعلى محرر الكود أضف الاستيرادات التالية

```
Imports System.Data.Linq
Imports System.Data.Linq.Mapping
```

سنضيف الآن Entity Class والذي عن عبارة عن فئة منظمة وفق جدول قاعدة بيانات ولإنشاء هذه الفئة نضيف الصفة Table قبل تعريف الفئة والت تمتلك الخاصية Name التي تحدد اسم الجدول في قاعدة البيانات ثم سيكون علينا إضافة الخصائص المناسبة لتمثل أعمدة الجدول ويتم تحديد الربط مع الأعمدة في قاعدة البيانات باستخدام الصفة Column والتي تمتلك عدة خصائص لتعريف العمود مثل IsPrimaryKey التي تمتلك قيمة منطقية تحدد فيما إذا كان العمود مفتاح أساسي أم لا والخاصية Storage التي تحدد اسم الحقل الخاص الذي سيخزن قيمة الخاصية – أضف الفئة التالية بعد تعريف الـ Module وقبل Sub Main والتي ستمثل الجدول Customers في قاعدة البيانات طبعا عرفت بعض الخصائص من أجل بعض الحقول الموجودة في الجدول هنا وليس جميع الحقول وذلك من اجل الاختصار هنا وإن قررت استخدام هذه الطريقة عمليا عليك تعريف الفئة بحيث تكون مطابقة تماما للجدول الذي تمثله

```
<Table(Name:="Customers")> _
Public Class Customer

    Private _CustomerID As String
    <Column(IsPrimaryKey:=True, Storage:="_CustomerID")> _
    Public Property CustomerID() As String
        Get
            Return Me._CustomerID
        End Get
        Set(ByVal value As String)
            Me._CustomerID = value
        End Set
    End Property

    Private _City As String
    <Column(Storage:="_City")> _
    Public Property City() As String
        Get
            Return Me._City
        End Get
        Set(ByVal value As String)
            Me._City = value
        End Set
    End Property

    Private _ContactName As String
    <Column(Storage:="_ContactName")> _
    Public Property ContactName() As String
        Get
            Return Me._ContactName
        End Get
        Set(ByVal value As String)
            Me._ContactName = value
        End Set
    End Property

End Class
```

سنحتاج إلى إنشاء اتصال مع قاعدة البيانات NorthWind التي سنستخدمها لذا سنحتاج إلى تعريف غرض DataContext والذي يعتبر القناة الرئيسية لقراءة البيانات وتخزينها في قاعدة البيانات – أدخل سطر الكود التالي في الإجراء Sub Main وذلك باعتبار أن قاعدة البيانات موجودة في الملف NORTHWND.MDF الموجود في المجلد D:\TEMP لدي عند تجربة هذا المثال

```
Dim db As New DataContext("D:\TEMP\NORTHWND.MDF")
```

ثم سنقوم بإنشاء كائن Table(of Customer) كي نستخدمه في الاستعلام من الجدول Customers أدخل الكود التالي مباشرة بعد الكود السابق

```
Dim Customers As Table(Of Customer) = _
          db.GetTable(Of Customer)()
```

أضف السطر التالي من الكود الذي سيطبع على نافذة الكونسول أوامر Sql التي سيتم تنفيذها على قاعدة البيانات

```
db.Log = Console.Out
```

سنكتب الآن استعلام Linq ليستعلم أي من الزبائن موجودين في لندن – أدخل كود الاستعلام التالي بعد الكود السابق مباشرة

```
Dim custQuery = From cust In Customers _
                Where cust.City = "London" _
                Select cust
```

وللحصول على نتائج الاستعلام سنستخدم حلقة For Each للدوران عبر نتائج الاستعلام كما رأينا في الدروس السابقة ثم طباعتها على نافذة الكونسول

```
Console.WriteLine("Number Of Records: " & custQuery.Count.ToString)

For Each CustObj In custQuery
     Console.WriteLine(CustObj.CustomerID.ToString & ", " & _
                       CustObj.City & ", " & CustObj.ContactName)
Next

Console.ReadLine()
```

وهذا هو الكود الكامل للمشروع

```
Imports System.Data.Linq
Imports System.Data.Linq.Mapping

Module Module1

     <Table(Name:="Customers")> _
     Public Class Customer

          Private _CustomerID As String
          <Column(IsPrimaryKey:=True, Storage:="_CustomerID")> _
          Public Property CustomerID() As String
               Get
                    Return Me._CustomerID
               End Get
               Set(ByVal value As String)
                    Me._CustomerID = value
               End Set
          End Property

          Private _City As String
          <Column(Storage:="_City")> _
```

```
        Public Property City() As String
            Get
                Return Me._City
            End Get
            Set(ByVal value As String)
                Me._City = value
            End Set
        End Property

        Private _ContactName As String
        <Column(Storage:="_ContactName")> _
        Public Property ContactName() As String
            Get
                Return Me._ContactName
            End Get
            Set(ByVal value As String)
                Me._ContactName = value
            End Set
        End Property

    End Class

    Sub Main()
        Dim db As New DataContext("D:\TEMP\NORTHWND.MDF")

        Dim Customers As Table(Of Customer) = _
                db.GetTable(Of Customer)()

        db.Log = Console.Out

        Dim custQuery = From cust In Customers _
                        Where cust.City = "London" _
                        Select cust

        Console.WriteLine("Number Of Records: " & custQuery.Count.ToString)

        For Each CustObj In custQuery
            Console.WriteLine(CustObj.CustomerID.ToString & ", " & _
                              CustObj.City & ", " & CustObj.ContactName)
        Next

        Console.ReadLine()

    End Sub

End Module
```

أمثلة على استعلامات Linq

في هذا الدرس سأورد بعض الأمثلة على استعلامات مختلفة كي نتعرف أكثر على أسلوب كتابة هذه الاستعلامات ولن أتطرق إلى طرق استخدام هذه الاستعلامات بما أننا رأينا كيف يمكن عمل ذلك من خلال الأمثلة الواردة في الدروس السابقة إما بالدوران باستخدام حلقة For ... Each أو بالإسناد المباشر لمتغير أو تحكم أو الإظهار في DataGridView

مثال بسيط على استخدام الدالات التجميعية في الاستعلام للحصول على القيمة العظمى لحقل يحتوي على مجموعة قيم

```
Dim Wod = Aggregate Wdt In ads.WorkingDate _
          Into Mdt = Max(Wdt.WorkingDate)
```

مثال آخر على استخدام الدالات التجميعية للحصول على مجموع ناتج عملية طرح حقلين

```
Dim TempBal = Aggregate AccBa In AccMov _
              Into Bal = Sum(AccBa.DebitAmmount - AccBa.CreditAmmount)
```

مثال بسيط آخر يستخدم Like في قسم Where في الاستعلام للحصول على مجموعة نتائج محددة

```
Dim AccMov = From AccBal In Accds.AccountMovements _
             Where AccBal.AccountNumber Like (AccNum & "*") _
             Select AccBal
```

مثال يستخدم الدالة ToLower في قسم Where للحصول على نتائج الاستعلام بحيث لا يتأثر شرط التصفية بحالة الأحرف

```
Dim Qts = From cu In Db.Customers _
          Where cu.Country.ToString.ToLower Like "po*".ToLower _
          Select cu
```

مثال بسيط آخر يستخدم Order By لترتيب النتائج تصاعديا

```
Dim ParentList = From ParLst In AccountsDataSet.Accounts _
                 Where ParLst.IsChildAccount = False _
                 Order By ParLst.AccountNumber _
                 Select ParLst
```

ومن أجل الترتيب تنازليا يصبح الاستعلام كما يلي وذلك بإضافة Descending بعد حقل الفرز في قسم Order By

```
Dim ParentList = From ParLst In AccountsDataSet.Accounts _
                 Where ParLst.IsChildAccount = False _
                 Order By ParLst.AccountNumber Descending _
                 Select ParLst
```

هذا المثال يقوم بإعادة تاريخ محتسب من تاريخ وفترة زمنية مخزنين في قاعدة بيانات مع استخدام Join لربط جدولين للحصول على القيم منهما

```
Dim Edat = From de In DsDesposits.Deposits _
           Join Dp In DsDesposits.InterestRates _
           On de.InterestID Equals Dp.InterestID _
           Select EndDate = DateAdd(DateInterval.Month, _
                                    intrst.First, DepDet.First.StartDate)
```

مثال آخر عن استعلام يستخدم Join لربط جدولين من أجل الاستعلام منهما ثم اختيار حقول محددة كنتيجة للاستعلام ثم استخدام الاستعلام داخل استعلام آخر من أجل الحصول على نتيجة إضافية ثم استخدام Join للاستعلام من نتيجة ربط استعلام وجدول للحصول على النتائج المرغوبة

```
Dim MovNam = From Acc In SampleDatabaseDataSet.ACCOUNTS _
             Join Mov In SampleDatabaseDataSet.ACCOUNTS_MOVEMENTS _
             On Acc.ACC_NUMBER Equals Mov.ACC_NUMBER _
             Select Mov.MOVMENT_DATE, Mov.ACC_NUMBER, Acc.ACC_NAME, _
             Mov.DEBIT_AMOUNT, Mov.CREDIT_AMOUNT

Dim AccBal = From AcBa In MovNam _
             Group By acc_number = AcBa.ACC_NUMBER _
             Into Balance = Sum(AcBa.DEBIT_AMOUNT - AcBa.CREDIT_AMOUNT)


Dim AccBalName = From ab In AccBal Join ac In SampleDatabaseDataSet.ACCOUNTS _
                 On ab.acc_number Equals ac.ACC_NUMBER _
                 Select ab.acc_number, ac.ACC_NAME, ab.Balance
```

هذا المثال يستخدم Join لإنشاء استعلام يعطينا النتائج نتيجة الاستعلام من أربعة جداول مختلفة

```
Dim ViewData = From Cu In DsDesposits.Customers _
             Join Dp In DsDesposits.Deposits On Dp.CustomerID _
             Equals Cu.CustomerID _
             Join DepPer In DsDesposits.InterestRates On DepPer.InterestID _
             Equals Dp.InterestID _
             Join CaDp In DsDesposits.CalculatedDepsits On CaDp.DepositID _
             Equals Dp.DepositID _
             Where CaDp.CalculationReason = 2 _
             And CaDp.CalculationDate >= FromDate _
             And CaDp.CalculationDate <= ToDate _
             Select Cu.Name, Dp.DepositNumber, Dp.DepositAmount, Dp.StartDate, _
             CaDp.CalculationDate, DepPer.DepositPreiod
```

هذا المثال فيه نقطتين الأولى بخصوص in المستخدمة في استعلامات sql للحصول على نتيجة من قائمة قيم فهي غير موجودة هنا لذا نقوم بوضع القيم في مصفوفة ثم نستخدم Contains في الاستعلام للحصول على نفس النتيجة والثانية هي أننا نستطيع استخدام نتيجة استعلام كدخل لاستعلام آخر

```
Dim Vlu() As Byte = {0, 3, 4, Nothing, 6}
Dim DepDet = From De In DsDesposits.Deposits _
             Where De.DepositID = Dr.DepositID _
             And Vlu.Contains(De.DepositStatus)

Dim Intrst = From Irs In DsDesposits.InterestRates _
             Where Irs.InterestID = DepDet.First.InterestID _
             Select Irs.DepositPreiod, Irs.InterestRate
```

مثال آخر يستخدم Join وطريقة المصفوفة معا

```
Dim Vlu() As Byte = {0, Nothing, 3, 4, 6}
mDep = From d In DsDesposits.Deposits _
       Join i In DsDesposits.InterestRates On d.InterestID Equals i.InterestID _
       Where DateAdd(DateInterval.Month, i.DepositPreiod, d.StartDate)_
              <= Now.Date _
       And Vlu.Contains(d.DepositStatus) _
       Select d.DepositID
```

هذا المثال يستخدم استعلامين متداخلين ففي قسم select في نهاية الاستعلام الأول استخدمنا الطريقة Except لاستثناء النتائج المعادة من الاستعلام الثاني الممرر كمحدد للطريقة Except

```
Deps = (From d In DsDesposits.Deposits _
        Join i In DsDesposits.InterestRates _
        On i.InterestID Equals d.InterestID _
        Where (d.StartDate <= New Date(Ryear, 12, 31) _
        And vlu.Contains(d.DepositStatus)) _
        And DateAdd(DateInterval.Month, i.DepositPreiod, d.StartDate)_
             >= New Date(Ryear, 12, 31) _
        Select d.DepositID).Except( _
        From ca In DsDesposits.CalculatedDepsits _
        Where ca.EndDate = New Date(Ryear, 12, 31) _
        Select ca.DepositID)
```

الاستعلامات المترجمة Compiled Queries

عندما يكون لدينا تطبيق يستخدم استعلامات متشابهة العديد من المرات يمكنك زيادة الأداء عبر ترجمة Compile الاستعلام مرة واحدة ثم تنفيذه العديد من المرات لاحقا عبر تمرير محددات مختلفة في كل مرة فقد يكون لديك تطبيق يقوم بالاستعلام عن جميع الزبائن الموجودين في مدينة محددة بحيث يتم تمرير اسم المدينة في وقت التنفيذ من قبل المستخدم وهنا تدعم Linq to Sql استخدام الاستعلامات المترجمة لهذا الغرض

حيث توفر لنا الفريمووورك الفئة CompiledQuery التي تزودنا بإمكانية الترجمة والتخزين المؤقت للاستعلامات من اجل الاستخدام وهذه الفئة متواجدة في مجال الأسماء System.Data.Linq وهي تمتلك خاصية وحيدة هي Expression التي تعيد الاستعلام كتعبير لمدا Lambda Expression وهي تمتلك بعض الطرق ولكن الطريقة الأهم والتي يتم استخدامها هنا هي الطريقة Compile التي تمتلك عدة أشكال محملة متشابهة

```
Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TResult) ( _
        query As Expression(Of Func(Of TArg0, TArg1, TResult)) _
         ) As Func(Of TArg0, TArg1, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TResult) ( _
        query As Expression(Of Func(Of TArg0, TArg1, TArg2, TResult)) _
         ) As Func(Of TArg0, TArg1, TArg2, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TArg3, TResult) ( _
        query As Expression(Of Func(Of TArg0, TArg1, TArg2, TArg3, TResult)) _
         ) As Func(Of TArg0, TArg1, TArg2, TArg3, TResult)


Public Shared Function Compile(Of TArg0 As DataContext, TResult) ( _
        query As Expression(Of Func(Of TArg0, TResult)) _
         ) As Func(Of TArg0, TResult)
```

حيث تمثل المحددات TArg نوع المحدد الممرر للمفوض Delegate عندما يتم تنفيذ المفوض المعاد من الطريقة Compile و المحدد TResult هو من النوع IEnumerable(T) المعاد عند تنفيذ المفوض المعاد من الطريقة Compile

في العديد من الحالات ربما ترغب في إعادة استخدام الاستعلامات خارج مجال مسار التنفيذ الحالي ففي هذه الحالات تكون عملية تخزين الاستعلامات المترجمة في متغيرات ساكنة Static Variables فكرة فعالة ففي الكود التالي توجد لدينا فئة Queries مصممة من أجل تخزين الاستعلامات المترجمة تستخدم للاستعلام من قاعدة بيانات NorthWind بطريقة Linq to Sql

```
Class Queries

    Public Shared CustomersByCity As _
        Func(Of NorthWindDataContext, String, IQueryable(Of Customer)) = _
            CompiledQuery.Compile(Function(db As NorthWindDataContext, _
            city As String) _
        From c In db.Customers Where c.City = city Select c)

    Public Shared CustomersById As _
        Func(Of NorthWindDataContext, String, IQueryable(Of Customer)) = _
            CompiledQuery.Compile(Function(db As NorthWindDataContext, _
            id As String) _
        db.Customers.Where(Function(c) c.CustomerID = id))

End Class
```

حيث يمكننا تعريف وظيفة تستخدم الاستعلام المترجم كما يلي

```
Public Function GetCustomersByCity(ByVal city As String) As  _
```

```
    IEnumerable(Of Customer)

        Return Queries.CustomersByCity(db, city)
End Function
```

ثم استخدام هذه الوظيفة لعرض نتائج الاستعلام في DataGridView كما يلي

```
Me.DataGridView1.DataSource = GetCustomersByCity("London").ToList
```

أو حتى الاستخدام المباشر للاستعلام المترجم بحيث نظهر نتيجة الاستعلام مباشرة في DataGridView

```
Me.DataGridView1.DataSource = Queries.CustomersByCity(db, "London").ToList
```

إضافة طرائق مخصصة لاستعلامات لينك Linq

يمكنك توسيع مجموعة الطرائق التي يمكنك استخدامها مع استعلامات لينك بإضافة وظائف موسعة للواجهة IEnumerable(T) فإضافة إلى العمليات التقليدية مثل الوسطي أو الحد الأقصى يمكننا مثلا إضافة دالة تجميعية لحساب قيمة وحيدة من سلسلة قيم كما يمكنك إنشاء دالة تشكل مرشح مخصص أو لتحويل سلسلة معينة من البيانات إلى سلسلة أخرى فعندما تقوم بتوسيع الواجهة IEnumerable(T) فأنت تضيف دالات مخصصة لأي مجموعة قابلة للتعداد.

فالمثال التالي يرينا كيفية إنشاء طريقة موسعة تدعى Median لحساب وسطي سلسلة أرقام من النوع Double

```
Imports System.Runtime.CompilerServices

Module LINQExtension

    ' Extension method for the IEnumerable(of T) interface.
    ' The method accepts only values of the Double type.
    <Extension()> _
    Function Median(ByVal source As IEnumerable(Of Double)) As Double
        If source.Count = 0 Then
            Throw New InvalidOperationException("Cannot compute median for an empty
set.")
        End If

        Dim sortedSource = From number In source _
                           Order By number

        Dim itemIndex = sortedSource.Count \ 2

        If sortedSource.Count Mod 2 = 0 Then
            ' Even number of items in list.
            Return (sortedSource(itemIndex) + sortedSource(itemIndex - 1)) / 2
        Else
            ' Odd number of items in list.
            Return sortedSource(itemIndex)
        End If
    End Function
End Module
```

حيث يمكنك استدعاء هذه الطريقة الموسعة من أي مجموعة قابلة للتعداد بنفس الطريقة التي تقوم بها باستدعاء الدالات التجميعية ويرينا المثال التالي كيفية استخدام الطريقة السابقة مع مصفوفة double

```
Dim numbers1() As Double = {1.9, 2, 8, 4, 5.7, 6, 7.2, 0}
Dim query1 = Aggregate num In numbers1 Into Median()
MsgBox("Double: Median = " & query1)
```

فالاستعلام السابق يجب أن ينتج القيمة 4.85 بحسب قيم الدخل

كما يمكن أن نضيف وظائف موسعة محملة Overloaded من أجل كل نوع من أنواع البيانات

```
' Integer overload
<Extension()> _
Function Median(ByVal source As IEnumerable(Of Integer)) As Double
    Return Aggregate num In source Select CDbl(num) Into med = Median()
End Function
```

ويكون استخدامها مماثلا

```
Dim numbers1() As Double = {1.9, 2, 8, 4, 5.7, 6, 7.2, 0}
Dim query1 = Aggregate num In numbers1 Into Median()
MsgBox("Double: Median = " & query1)

Dim numbers2() As Integer = {1, 2, 3, 4, 5}
```

```
Dim query2 = Aggregate num In numbers2 Into Median()
MsgBox("Integer: Median = " & query2)
```

كما يمكننا إضافة طريقة موسعة محملة تقبل أغراضا عامة Generic Objects وهي تأخذ مفوضا Delegate كمحدد وتستخدمه لتحويل سلسة الأغراض من النوع العام للنوع المحدد ويرينا الكود التالي طريقة محملة لـ Median تأخذ Func(T, TResult) كمحدد مفوض وهذا المفوض يأخذ غرض من نوع عام T ويعيد غرض من النوع Double

```
' Generic overload.
<Extension()> _
Function Median(Of T)(ByVal source As IEnumerable(Of T), _
                      ByVal selector As Func(Of T, Double)) As Double
    Return Aggregate num In source Select selector(num) Into med = Median()
End Function
```

حيث يمكنك استخدام الطريقة Median من اجل سلسلة من الأغراض من أي نوع إذا لم يكن للنوع وظيفته المحملة فيجب عليك عندها تمرير محدد مفوض Delegate Parameter كما يمكنك استخدام تعابير لمدا Lambda Expressions لهذا الغرض كما يمكنك في فيجول بايزيك فقط استخدام قسم Aggregate أو Group By بدلا من استدعاء الطريقة حيث يمكنك تمرير أي قيمة أو تعبير ضمن مجال القسم

ويرينا المثال التالي كيفية استدعاء الطريقة Median من أجل integer أو String فمن أجل النصوص يحتسب طول النصوص في المصفوفة حيث نرى كيفية تمرير Func(T, TResult) للطريقة Median من أجل كل حالة

```
Dim numbers3() As Integer = {1, 2, 3, 4, 5}

' You can use num as a parameter for the Median method
' so that the compiler will implicitly convert its value to double.
' If there is no implicit conversion, the compiler will
' display an error message.

Dim query3 = Aggregate num In numbers3 Into Median(num)
MsgBox("Integer: Median = " & query3)
Dim numbers4() As String = {"one", "two", "three", "four", "five"}

' With the generic overload, you can also use numeric properties of objects.

Dim query4 = Aggregate str In numbers4 Into Median(str.Length)
MsgBox("String: Median = " & query4)

' This code produces the following output:
' Integer: Median = 3
' String: Median = 4
```

يمكننا توسيع الواجهة IEnumerbale(T) بطريقة استعلام مخصصة تعيد سلسلة من القيم وفي هذه الحالة يجب علينا إعادة مجموعة من النوع IEnumerable(T) حيث يمكن استخدام طريقة مماثلة لما ذكر من اجل ترشيح أو تحويل بيانات إلى سلسلة من القيم ويرينا المثال التالي كيفية إنشاء طريقة موسعة تدعى AlternateElements تعيد جميع العناصر الأخرى في المجموعة بدءا من العنصر الأول

```
' Extension method for the IEnumerable(of T) interface.
' The method returns every other element of a sequence.
<Extension()> _
Function AlternateElements(Of T)(ByVal source As IEnumerable(Of T)) _
                                 As IEnumerable(Of T)
    Dim list As New List(Of T)
    Dim i = 0
    For Each element In source
        If (i Mod 2 = 0) Then
            list.Add(element)
        End If
```

```
        i = i + 1
    Next
    Return list
End Function
```

حيث يمكنك استدعاء هذه الطريقة الموسعة من أجل أي مجموعة قابلة للتعداد تماما كما تفعل عندما تستدعي أي طريقة أخرى من الواجهة IEnumerbale(T) كما نرى في الكود

```
Dim strings() As String = {"a", "b", "c", "d", "e"}
Dim query = strings.AlternateElements()
For Each element In query
    MsgBox(element)
Next

' This code produces the following output:
' a
' c
' e
```

القسم الثامن - الفئات والواجهات ومجالات الأسماء

ويضم المواضيع التالية:

- تجزئة الفئة أو التركيب على عدة ملفات
- Overriding WndProc
- الواجهات Interfaces
- تحقيق الواجهة IEnumerable
- إدارة المصادر والواجهة IDisposable
- Using Generics with Interfaces
- نظرة ضمن مجال الأسماء MY
- كيف تقوم بإضافة إجراءاتك الخاصة إلى مجال الأسماء My
- كيف تستطيع إطلاق أحداثك الخاصة RaiseEvent Tutorial
- الطرائق المُوَسِّعة Extension Methods
- الطريقة Main
- التحميل الزائد للمعاملات Operators Overloading
- إنشاء مكتبة تضيف وظائف جديدة للتحكمات الموجودة بدون استخدام الوراثة
- توسيع مجال الأسماء My باستخدام My Extensibility
- جعل صندوق النصوص يقبل العمليات الحسابية بدون استخدام الخاصية Text ودوال تحويل الأنواع

تجزئة الفئة أو التركيب على عدة ملفات

أنشئ مشروعا جديدا و أضف إليه ملفين Code File

في الملف الأول أدخل الكود التالي الذي هو عبارة عن فئة اسمها Person

```
Public Class Person

   Public Property FirstName() As String
       Get
           Return _FirstName
       End Get
       Set(ByVal value As String)
           _FirstName = value.Trim
       End Set
   End Property

   Public Property LastName() As String
       Get
           Return _LastName
       End Get
       Set(ByVal value As String)
           _LastName = value.Trim
       End Set
   End Property

   Public Sub New()
       _FirstName = "John"
       _LastName = "Doe"
   End Sub

End Class
```

الآن في الملف الثاني أدخل الكود التالي الذي هو عبارة عن تتمة الفئة person نفسها

```
Partial Public Class Person

   Private _FirstName As String
   Private _LastName As String

   Public Sub New(ByVal FirstName As String, ByVal LastName As String)
       _FirstName = FirstName.Trim
       _LastName = LastName.Trim
   End Sub

   Public ReadOnly Property FullName() As String
       Get
           Return _FirstName & " " & _LastName
       End Get
   End Property
```

```
End Class
```

لاحظ في الملف الثاني قبل تعريف الفئة استخدام الكلمة Partial وهي التي تمكننا من تجزئة الفئة Class أو التركيب Structure إلى عدة ملفات

```
Partial Public Class Person
```

يستخدم فيجول بايزيك تعريف الفئة الجزئية لفصل الكود المولد تلقائيا من الكود المكتوب من قبل المستخدم على ملفات منفصلة. فعلى سبيل المثال مصمم النماذج يحدد فئات Class جزئية للتحكمات مثل النموذج .From وعليك ألا تعدل الكود المولد تلقائيا لهذه التحكمات. وهذا مثال آخر

```
Partial Public Class sampleClass
   Public Sub sub1()
   End Sub
End Class

Partial Public Class sampleClass
   Public Sub sub2()
   End Sub
End Class
```

ويبقى استخدام الفئة كما هو كما لو كان ضمن ملف واحد كما كنا نفعل سابقا

Overriding WndProc

يرسل نظام التشغيل – الويندوز – جميع أنواع الرسائل للتطبيقات والتي تخبرها عن التغييرات في بيئة النوافذ. وهذه الرسائل تخبر النموذج بالقيام بعدة أشياء كإعادة الرسم، النقل، إعادة التحجيم، الإخفاء، التصغير، والإغلاق بالإضافة إلى الاستجابة للتغييرات في بيئة النوافذ أو القيام بأي شئ آخر متعلق بالنوافذ. ويكون لجميع تطبيقات النوافذ إجراء يتم إطلاقه للاستجابة لتلك الرسائل ويدعى هذا الإجراء في العادة WindowProc ويعالج فيجول بايزيك دوت نيت هذه الرسائل في إجراء يدعى WndProc الذي يمكنك تجاوزه لعمل أشياء مخصصة عند استقبال برنامجك لرسائل معينة.

فمثلا الكود التالي يظهر كيف يضمن البرنامج إبقاء النافذة لنفس نسب قياسات الواجهة بحيث نقوم بتجاوز الإجراء WndProc الخاص بالنموذج والبحث عن الرسالة WM_SIZING التي تستقبل محددات تتضمن الحافة التي يقوم المستخدم بجرها لتغيير حجم النموذج وتركيب من النوع Rect يعطي للنموذج مكانه وحجمه الجديدين ويبدأ الكود بتحديد التركيب Rect ثم يعلن عن إجراء تجاوز للإجراء WndProc والذي يحدد بعض الثوابت والمتغيرات الساكنة التي تحمل القيم الأساسية للنموذج ثم يحدد الإجراء WndProc نوع الرسالة التي يقوم بمعالجتها فإن كانت WM_SIZING تستخدم الوظيفة PtrToStructure لنسخ m.LParam إلى التركيب Rect الذي يتم استخدامه لحساب العرض والطول الجديدين للنموذج والنسبة بينهما. فإن كانت هذه هي المرة الأولى التي يتم تنفيذ WndProc فيها فتكون بذلك قيمة المتغير الساكن fixed_aspect_ratio مساوية للصفر وعندما يرى أن قيمته مساوية للصفر يقوم بتخزين نسبة الطول والعرض الحاليين للنموذج في ذلك المتغير ثم يقوم WndProc بتحديد فيما إذا كانت النسبة الخاصة بالنموذج مختلفة عن القيمة الأصلية فإن تم تغييرها يقرر أي بعد (طول أو عرض) سيتم حفظه فإن كان المستخدم يقوم بالسحب من إحدى الزوايا يقوم الإجراء بحساب أي البعدين (طول أو عرض) هو الأكبر ثم يقوم بحساب قيمة البعد الآخر الذي يحقق نسبة الطول للعرض الخاصة بالنموذج. وإن كان المستخدم يقوم بسحب أحد الأطراف يقوم البرنامج بالحفاظ على العرض الجديد ويقوم بحساب الارتفاع المناسب حسب النسبة وكذلك إن كان يقوم بالسحب باستخدام إحدى الحافتين العلوية أو السفلية يقوم بتثبيت الارتفاع ويقوم بحساب العرض الملائم حسب النسبة ثم يقوم البرنامج بتقرير فيما إذا كان يجب عليه نقل النموذج بحيث يحرك الطرف الذي يقوم المستخدم بسحبه فمثلا إن كان المستخدم يسحب الزاوية اليسارية السفلى فالبرنامج يغير قيم اليسار والأسفل بحيث تبقى الزاوية العلوية اليمنى ثابتة ثم يقوم WndProc باستدعاء Marshal.StructureToPtr لينسخ التركيب Rect مجددا إلى m.LParam وأخيرا يقوم WndProc باستدعاء MyBase.WndProc ليترك المجال للإجراء WndProc الأب لاستخدام القيم الجديدة لتغيير حجم النموذج. واستدعاء الإجراء WndProc الخاص بالأب هام جدا فإن لم يقم البرنامج باستدعائه لكل رسالة لم يتم معالجتها بشكل كامل فإن تلك الرسالة لن يتم معالجتها وبالنتيجة فالنافذة لن تقوم بمعالجة جميع الرسائل الخاصة بها مما ينتج عن ذلك عدة مشاكل مختلفة كعدم قدرتها على إعادة رسم نفسها أو التحريك أو إظهار القوائم أو غيرها من الأمور الخاصة بالنموذج

```vb
Imports System.Runtime.InteropServices

Public Class Form1

    Public Structure Rect
        Public left As Integer
        Public top As Integer
        Public right As Integer
        Public bottom As Integer
    End Structure

    Protected Overrides Sub WndProc(ByRef m As System.Windows.Forms.Message)
        Const WM_SIZING As Long = &H214
        Const WMSZ_LEFT As Integer = 1
        Const WMSZ_RIGHT As Integer = 2
        Const WMSZ_TOP As Integer = 3
        Const WMSZ_TOPLEFT As Integer = 4
        Const WMSZ_TOPRIGHT As Integer = 5
        Const WMSZ_BOTTOM As Integer = 6
        Const WMSZ_BOTTOMLEFT As Integer = 7
        Const WMSZ_BOTTOMRIGHT As Integer = 8
        Static fixed_aspect_ratio As Double = 0
        Dim new_aspect_ratio As Double
        If m.Msg = WM_SIZING And m.HWnd.Equals(Me.Handle) Then
            ' Turn the message's lParam into a Rect.
```

```
Dim r As Rect
r = DirectCast( _
Marshal.PtrToStructure(m.LParam, GetType(Rect)), _
Rect)
' Get the current dimensions.
Dim wid As Double = r.right - r.left
Dim hgt As Double = r.bottom - r.top
' Get the new aspect ratio.
new_aspect_ratio = hgt / wid
' The first time, save the form's aspect ratio.
If fixed_aspect_ratio = 0 Then
    fixed_aspect_ratio = new_aspect_ratio
End If
' See if the aspect ratio is changing.
If fixed_aspect_ratio <> new_aspect_ratio Then
    ' To decide which dimension we should preserve,
    ' see what border the user is dragging.
    If m.WParam.ToInt32 = WMSZ_TOPLEFT Or _
     m.WParam.ToInt32 = WMSZ_TOPRIGHT Or _
     m.WParam.ToInt32 = WMSZ_BOTTOMLEFT Or _
     m.WParam.ToInt32 = WMSZ_BOTTOMRIGHT Then

        ' The user is dragging a corner.
        ' Preserve the bigger dimension.
        If new_aspect_ratio > fixed_aspect_ratio Then
            ' It's too tall and thin. Make it wider.
            wid = hgt / fixed_aspect_ratio
        Else
            ' It's too short and wide. Make it taller.
            hgt = wid * fixed_aspect_ratio
        End If
    ElseIf m.WParam.ToInt32 = WMSZ_LEFT Or _
             m.WParam.ToInt32 = WMSZ_RIGHT Then

        ' The user is dragging a side.
        ' Preserve the width.
        hgt = wid * fixed_aspect_ratio
    ElseIf m.WParam.ToInt32 = WMSZ_TOP Or _
             m.WParam.ToInt32 = WMSZ_BOTTOM Then

        ' The user is dragging the top or bottom.
        ' Preserve the height.
        wid = hgt / fixed_aspect_ratio
    End If
    ' Figure out whether to reset the top/bottom
    ' and left/right.
    ' See if the user is dragging the top edge.
    If m.WParam.ToInt32 = WMSZ_TOP Or _
         m.WParam.ToInt32 = WMSZ_TOPLEFT Or _
         m.WParam.ToInt32 = WMSZ_TOPRIGHT Then

        ' Reset the top.
        r.top = r.bottom - CInt(hgt)
    Else
        ' Reset the bottom.
        r.bottom = r.top + CInt(hgt)
    End If
    ' See if the user is dragging the left edge.
    If m.WParam.ToInt32 = WMSZ_LEFT Or _
         m.WParam.ToInt32 = WMSZ_TOPLEFT Or _
         m.WParam.ToInt32 = WMSZ_BOTTOMLEFT Then

        ' Reset the left.
```

```
                r.left = r.right - CInt(wid)
            Else
                ' Reset the right.
                r.right = r.left + CInt(wid)
            End If
            ' Update the Message object's LParam field.
            Marshal.StructureToPtr(r, m.LParam, True)
        End If
    End If
    MyBase.WndProc(m)
End Sub

End Class
```

في فيجول بايزيك 6 والنسخ السابقة له يمكن للبرنامج تثبيت إجراء WindowProc خاص لعمل نفس العمليات بصعوبة وتدعى هذه العملية بـ subclassing ويعتبر الاسم غير موفق لأن البرمجة غرضية التوجه تستخدم نفس الاسم subclassing للإشارة إلى إنشاء فئة من أخرى كما نفعل عند استخدامنا للعبارة Inherits وتكون عملية تجاوز WndProc في فيجول بايزيك دوت نيت أسهل بكثير وأكثر أمنا من الطريقة المستخدمة في فيجول بايزيك 6 كما تلاحظ في المثال ولكنك مازلت بحاجة لبعض الخدع لتحويل IntPtr المخزن في m.LParam من وإلى تراكيب مناسبة. كما يكون عليك تقرير أي من الرسائل ستقوم باعتراضها وأي من قيم m.LParam و m.WParam تأخذ وكيف يمكنك التأثير عليهم بأمان. وإحدى الطرق لتعلم هذه الرسائل هو إدراج الإجراء WndProc التالي ثم القيام بالعمل الذي تريد دراسته (تغيير حجم النموذج على سبيل المثال)

```
Protected Overrides Sub WndProc(ByRef m As System.Windows.Forms.Message)
    Debug.Print(m.ToString)
    MyBase.WndProc(m)
End Sub
```

والسطر التالي يبين النتيجة من أجل الرسالة WM_SIZING المرسلة إلى النموذج عندما يقوم المستخدم بتغيير حجمه وعلى الأقل يظهر اسم الرسالة WM_SIZING وقيمتها الست عشرية 0x214

msg=0x214 (WM_SIZING) hwnd=0x30b8c wparam=0x2 lparam=0x590e29c result=0x0

والبحث عن اسم الرسالة في موقع مايكروسوفت ومواقع البرمجة الأخرى يعطيك في العادة معلومات أخرى تحتاج لمعرفتها مثل معنى قيم m.WParam و m.LParam كما يجدر ملاحظة أن Form class يرث الإجراء WndProc من Control class والتي ترثه بقية التحكمات على النموذج بدورها و الذي يعني أنك تستطيع تجاوز الإجراء WndProc بأي تحكم منها وبذلك يمكنك التحكم بتصرفاتها. فمثلا ترينا قطعة الكود التالية كيف تعمل الفئة NoCtxMnuTextBox والمشتقة من التحكم TextBox بحيث يقوم الإجراء WndProc الخاص بها بتفحص الرسالة WM_CONTEXTMENU ثم يستدعي الإجراء الخاص بالفئة الأب من أجل جميع الرسائل الأخرى وبالقيام بإفشال معالجة الرسالة WM_CONTEXTMENU نمنع إظهار القائمة المنبثقة الخاصة بالتحكم والتي تضم أوامر مثل النسخ واللصق عندما تنقر بزر الماوس اليميني عليها

```
Public Class NoCtxMnuTextBox
    Inherits System.Windows.Forms.TextBox
    Protected Overrides Sub WndProc(ByRef m As System.Windows.Forms.Message)
        Const WM_CONTEXTMENU As Integer = &H7B
        If m.Msg <> WM_CONTEXTMENU Then
            MyBase.WndProc(m)
        End If
    End Sub
End Class
```

الواجهات Interfaces

أفترض بمن يتابع معي أن تكون له دراية عن الفئات وإنشائها والتعامل معها

الواجهة هي نوع مرجعي تستخدمه الأنواع الأخرى لضمان أنها تدعم عملية معينة وهي تحدد عناصر معينة يجب أن تتضمنها الفئات التي تحقق هذه الواجهات وهي تحتوي على طرائق وخصائص وعناصر أحداث تماما كالفئات

الصيغة العامة

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] _
Interface name [ ( Of typelist ) ]
    [ Inherits interfacenames ]
    [ [ modifiers ] Property membername ]
    [ [ modifiers ] Function membername ]
    [ [ modifiers ] Sub membername ]
    [ [ modifiers ] Event membername ]
    [ [ modifiers ] Interface membername ]
    [ [ modifiers ] Class membername ]
    [ [ modifiers ] Structure membername ]
End Interface
```

وكما نرى من الصيغة العامة فيسبق التصريح عن الواجهة واصفات Attributes وكلمات تحديد المجال مثل Public أو الكلمة Shadows التي تعني أن هذه الواجهة تعيد تعريف واجهة موجودة وبنفس الاسم كما يمكن وراثة واجهة من أخرى تماما كالفئات وهي تحتوي على نفس العناصر الممكن احتوائها ضمن الفئات من وظائف ودالات وخصائص ... الخ ولكنها تحدد تعريف هذه الوظائف والخصائص فقط بدون الكود الذي يحدد عملها ويجب على أي فئة تحقق واجهة معينة أن توفر الكود العملي لكافة العناصر الموجودة ضمن هذه الواجهة ويمكن تعريف الواجهة على مستوى Namespace أو Module أي يجب أن يكون تعريفها عاما وغير محصور ضمن إجراء معين كما يمكننا تعشيش الواجهات بحيث أن أية واجهة ممكن أن تتضمن واجهة أخرى كما يمكن تحديد خاصية افتراضية باستخدام الكلمة Default ولا يمكن استخدام محددات الوصول مثل Public أو Private عند التصريح عن عناصر الواجهة ولكن يمكن استخدام Overloads أو Shadows وعندما تستخدم واجهة ضمن فئة يتم الإعلان عن عناصر هذه الفئة باستخدام محدد الوصول Public افتراضيا الأمر الذي يمكنك تغييره لاحقا على مستوى تلك الفئة كما لا يمكن التصريح عن متغيرات ضمن الواجهة وعند تسمية الواجهات يفضل أن يبدأ الاسم دوما بالحرف I

لنرى الآن بعض الأمثلة عن الواجهات

يمكن أن نعرف واجهة لأشخاص تحتوي على بعض الخصائص كما يلي

```
Interface IPerson
    Property Name() As String
    Property Birthdate() As Date
    ReadOnly Property Age() As Integer
End Interface
```

أو يمكننا تعريف واجهة لبعض العمليات الحسابية

```
Interface ISomeMath
    Function AddNumbers(ByVal a As Integer, ByVal b As Integer) As Integer
    Function AddNumbers(ByVal a As Double, ByVal b As Double) As Double
    Function Multiply(ByVal a As Integer, ByVal b As Integer) As Double
End Interface
```

كما يمكن أن نعرف واجهة لدفتر الهواتف ترث واجهة الأشخاص كما يلي

```
Interface IPhonebook
    Inherits IPerson

    Property Phone() As String
    Property Address() As String
    Sub ShowInformations()
End Interface
```

الآن إن كانت لدينا فئة للهواتف نريد منها أن تحقق الواجهة Phonebook نستخدم الكلمة Implements تماما بعد التصريح عن تلك الفئة وسنرى أن بيئة التطوير ستضيف الهيكل الأساسي لعناصر تلك الواجهة إلى الفئة الجديدة

```
Public Class Phones
    Implements IPhonebook

    Public ReadOnly Property Age() As Integer Implements IPerson.Age
        Get

        End Get
    End Property

    .......

End Class
```

كما يمكن استخدام أكثر من واجهة ضمن الفئة الواحدة

```
Class SomeTest
    Implements IPerson
    Implements ISomeMath

    Public ReadOnly Property Age() As Integer Implements IPerson.Age
        Get

        End Get
    End Property

    .......


    Public Function AddNumbers(ByVal a As Double, ByVal b As Double) As Double _
                Implements ISomeMath.AddNumbers

    End Function

   .......


End Class
```

ويبقى عليك كتابة الكود المناسب لتلك العناصر بما يتناسب مع وظيفة الفئة التي تعمل عليها لاحظ تعريف جميع العناصر المضافة باستخدام محدد الوصول Public في هذه المرحلة

وهذا مثال على تعشيش الواجهات بداخل بعضها

```
Interface IPhonebook
```

```
    Interface IPersone
        Property Name() As String
        Property Birthdate() As Date
        ReadOnly Property Age() As Integer
    End Interface

    Property Phone() As String
    Property Address() As String
    Sub ShowInformations()
    Event SomeEvent(ByVal a As Int16)
End Interface
```

واستخدامها ضمن الفئة

```
Class test
    Implements IPhonebook
    Implements IPhonebook.IPersone

    .......

End Class
```

سؤال:

الحقيقة لا أجد الواجهات سوى أنها تجبر مستخدمها على تعريف أعضاء الواجهة (خصائص أو دوال أو وظائف) بداخل الفئة . بعبارة أخرى تستورد هيكل فئة من دون أي كود !!

فماذا سأستفيد من هذا الأمر ؟

الجواب

إذا تنقلت في بعض الفئات والواجهات في msdn ستجد أن مايكروسوفت تلزمك باستخدام واجهة معينة لأداء غرض معين (أي ضمان أن فئتك تتضمن عناصر معينة لازمة لإتمام المهمة التي أنت بصدد القيام بها) فمثلا عندما تستخدم عبارة Using للتصريح عن متغير يجب أن تكون الفئة التي نعرف هذا المتغير من نوعها محققة للواجهة IDisposable وذلك لأن Using تحتاج إلى عناصر معينة أن تكون متوفرة في المتغير الذي تعرفه وهذه العناصر توفرها للفئة الواجهة IDisposable

تحقيق الواجهة IEnumerable(Of T)

يتم تعريف الواجهة IEnumerable(T) بواسطة فئات تستطيع إعادة سلسلة من القيم بشكل متسلسل قيمة قيمة وتكمن الفائدة من إعادة البيانات بهذه الطريقة هي أنه لا يجب عليك تحميل مجموعة كاملة من البيانات في الذاكرة من أجل العمل عليها فأنت تستخدم ذاكرة كافية لتحميل عنصر واحد من البيانات ويمكن استخدام الفئات التي تحقق الواجهة IEnumerbale(T) مع حلقات For … Next أو مع استعلامات لينك Linq

فإن كان على تطبيق معين القراءة من ملف نصي كبير ويعيد أسطر الملف التي توافق معيار بحث معين فيستخدم التطبيق استعلامات لينك للعودة بالأسطر التي تطابق معيار بحث معين وللاستعلام عن محتويات الملف بواسطة استعلام لينك يجب على التطبيق تحميل محتويات الملف ضمن مصفوفة أو مجموعة ولكن عملية تحميل الملف بأكمله ضمن مصفوفة أو مجموعة تستهلك قدرا كبيرا من الذاكرة وبدلا عن ذلك يمكن لاستعلام لينك الاستعلام عن محتويات الملف باستخدام فئة قابلة للتعداد Enumerable Class تعيد فقط تلك القيم التي تطابق شرط البحث فالاستعلامات التي تعيد بضعة قيم مطابقة تستهلك قدرا أقل بكثير من الذاكرة

ويمكنك إنشاء فئة تحقق الواجهة IEnumerable(T) لتقدم البيانات المصدرية كبيانات قابلة للتعداد وفئتك التي تحقق الواجهة IEnumerable(T) ستحتاج إلى فئة ثانية تحقق الواجهة IEnumerator(T) للدوران حول البيانات المصدرية بحيث تمكنك هاتان الفئتان من إعادة عناصر البيانات بشكل متسلسل من نوع محدد وفي هذا المثال سنقوم بإنشاء فئتان الأولى تحقق الواجهة IEnumerbale(Of String) والثانية تحقق الواجهة IEnumerator(Of String) بحيث تستخدمان للقراءة من ملف نصي سطر واحد في كل مرة

أنشئ مشروعا جديدا من النوع Class Library وقم بتسميته StreamReaderEnumerable ثم من Solution Explorer قم بإعادة تسمية الملف Class1.vb إلى StreamReaderEnumerable.vb وإن سألتك بيئة التطوير اقبل تغيير اسم الفئة إلى StreamReaderEnumerable ثم انقر بزر الفأرة اليميني على المشروع StreamReaderEnumerable واختر Add ثم New item ثم Class وقم بتسمية الملف StreamReaderEnumerator.vb

افتح الملف StreamReaderEnumerable.vb وفي القسم العام بعد تعريف الفئة ادخل

```
Implements IEnumerable(Of String)
```

ثم اضغط Enter فيقوم فيجول بايزيك تلقائيا بإضافة العناصر الضرورية لتحقيق الواجهة IEnumerable(Of String) interface إلى كودك وسنقوم الآن بإضافة باني عام للفئة يأخذ محدد واحد هو مسار الملف

```
Private _filePath As String

Public Sub New(ByVal filePath As String)
    _filePath = filePath
End Sub
```

سنقوم الآن بإضافة الكود المناسب للوظيفة GetEnumerator بحيث تعيد تواجدا جديدا للفئة StreamReaderEnumerator ويمكن جعل تعريف هذه الوظيفة خاصا لأنك تريد عرض عناصر الواجهة IEnumerable(Of String) فقط اجعل كود الوظيفة مطابقا للتالي

```
Public Function GetEnumerator() _
    As System.Collections.Generic.IEnumerator(Of String) _
    Implements System.Collections.Generic.IEnumerable(Of String).GetEnumerator

    Return New StreamReaderEnumerator(_filePath)
End Function

Public Function GetEnumerator1() _
    As System.Collections.IEnumerator _
    Implements System.Collections.IEnumerable.GetEnumerator

    Return Me.GetEnumerator()
End Function
```

افتح الملف StreamReaderEnumerator.vb وفي القسم العام للفئة StreamReaderEnumerator أدخل السطر التالي ثم اضغط Enter

```
Implements IEnumerator(Of String)
```

فيتم إضافة العناصر الضرورية لتحقيق الواجهة IEnumerator(Of String) لكودك أضف الكود التالي الذي سيشكل باني الفئة التي ستقوم بفتح الملف وتنفيذ عمليات الدخل والخرج عليه بحيث يأخذ محددا وحيدا هو مسار الملف

```
Private _sr As IO.StreamReader

Public Sub New(ByVal filePath As String)
    _sr = New IO.StreamReader(filePath)
End Sub
```

الخصائص الحالية لـلواجهات IEnumerator(Of String) و IEnumerator تعيد العنصر الحالي من الملف النصي كنص ويمكن جعل تعريف الخاصية Current خاصا لأنه عليك عرض خصائص الواجهة IEnumerable(Of String) فقط الآن اجعل كود الخاصية Current مطابقا للتالي

```
Private _current As String

Public ReadOnly Property Current() As String _
    Implements IEnumerator(Of String).Current

    Get
        If _sr Is Nothing OrElse _current Is Nothing Then
            Throw New InvalidOperationException()
        End If

        Return _current
    End Get
End Property

Private ReadOnly Property Current1() As Object _
    Implements IEnumerator.Current

    Get
       Return Me.Current
    End Get
End Property
```

تتنقل الطريقة MoveNext للواجهة IEnumerator للعنصر التالي في الملف النصي وتحدث القيمة المعادة من الخاصية Current وإن لم يعد يوجد أية عناصر أخرى تعيد الطريقة MoveNext القيمة False وإلا فسوف تعيد القيمة True الآن اجعل كود MoveNext كما يلي

```
Public Function MoveNext() As Boolean _
    Implements System.Collections.IEnumerator.MoveNext

    _current = _sr.ReadLine()
    If _current Is Nothing Then Return False
    Return True
End Function
```

وتوجه الطريقة Reset للواجهة IEnumerator نقطة التكرار بالانتقال إلى نقطة البداية للملف النصي وتفرغ قيمة الخاصية CurrentItem

أجعل كود الطريقة Reset كما يلي

```
Public Sub Reset() _
   Implements System.Collections.IEnumerator.Reset

    _sr.DiscardBufferedData()
    _sr.BaseStream.Seek(0, IO.SeekOrigin.Begin)
    _current = Nothing
End Sub
```

وتضمن الطريقة Dispose للواجهة IEnumerator أن جميع المصادر الغير مدارة سيتم تحريرها قبل تدمير التكرار فمقبض الملف الذي يستخدم من قبل الغرض StreamReader هو موارد غير مدارة ويجب أن يتم إغلاقه قبل أن يتم تدمير تواجد التكرار استبدل الكود الذي ولد من أجل الطريقة Dispose بالكود التالي

```
Private disposedValue As Boolean = False

Protected Overridable Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposedValue Then
        If disposing Then
            ' Dispose of managed resources.
        End If
        _current = Nothing
        _sr.Close()
        _sr.Dispose()
    End If

    Me.disposedValue = True
End Sub

Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub

Protected Overrides Sub Finalize()
    Dispose(False)
End Sub
```

من أجل تجربة الفئات التي قمنا بإنشائها قم بإضافة مشروع جديد من النوع Windows Application إلى Solution StreamReaderEnumerable ثم قم بإضافة مرجع لمكتبة الفئات StreamReaderEnumerable وذلك بالنقر بزر الفأرة الأيمن على المشروع الجديد ثم اختيار Add Reference ومن صفحة Projects اختر مكتبة الفئات المذكورة

أضف على النموذج ListBox و Button واجعل كود النموذج كما يلي

```
Imports StreamReaderEnumerable

Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click
        Dim adminRequests = _
    From line In New
StreamReaderEnumerable.StreamReaderEnumerable("c:\ipconfig.txt") _
    Where line.Contains("Display")

        Me.ListBox1.Items.Clear()
        Me.ListBox1.Items.AddRange(adminRequests.ToArray)
    End Sub

End Class
```

حيث استخدمنا الفئة StreamReaderEnumerable في استعلام لينك يجلب السطور من الملف c:\ipconfig.txt التي تحتوي على الكلمة Display

إدارة المصادر والواجهة IDisposable

يقدم Visual Basic .net انعطافة جديدة في إدارة المصادر. حيث يوفر الـ CLR تقنية لإدارة المصادر تعرف باسم جمع النفايات Garbage Collection وذلك لتحرير المطور من معظم مهام إدارة الذاكرة. ولكن هذا لا يأتي بدون ثمن ولا يحررك بشكل كامل من التعامل مع إدارة الذاكرة. ففي بعض الحالات يجبرك جامع النفايات على أن تكون مدركا أكثر بخصوص العناصر التي يجب أن تقوم بالتخلص منها يدويا والتأكد من أن المصادر قد تم تحريرها بطريقة مناسبة. ولكن في أغلب الحالات لن تضطر للقلق حول متى يتم تدمير الأشياء.

في Visual Basic 6.0 عندما تصبح جميع المتغيرات التي تشير إلى شئ Object معين معينة إلى Nothing عندها يتم تحرير الذاكرة المرتبطة بهذه المتغيرات بشكل فوري. ويأخذ الـ CLR الآن هذه المسؤولية حول إدارة المصادر مما يمكن المطورين من التركيز على تحسين برامجهم عوضا عن كتابة الكثير من شيفرة إدارة الذاكرة.

وهذا يتطلب تغييرات في ممارسات كتابة الشيفرة. فمثلا عندما تضبط متغير ما إلى Nothing لا يتم تدمير الشئ Object مباشرة ولكن سوف يتم تدميره لاحقا وهذا ما يحدده جامع النفايات. وفي أغلب الحالات لا يجب أن يؤثر هذا على تطبيقك ولكن في بعض الحالات كبرنامج يقع تحت ضغط قلة مصادر النظام المتاحة يمكن أن يشكل معضلة حقيقية من ناحيتي الأداء والمرونة. ففي الحالات التي يكون فيها الأداء أساسي يكون من أول اهتماماتك إفراغ مصادر النظام فور فراغك من استخدامها.

أبق في ذهنك أن دورات جامع النفايات مكلفة. والفكرة تكمن في منع جامع النفايات من العمل بشكل متكرر كثيرا فكلما قمت بالتنظيف خلفك كلما تركت عمل أقل لجامع النفايات لعمله وبذلك تقوم بتحسين أداء برنامجك.

جامع النفايات The Garbage Collector

إذا كان التطبيق يستخدم مصادر غير مدارة Unmanaged Resources كتحديد ذاكرة خاصة أو مؤشرات الملفات أو عناصر أخرى من مصادر النظام فإن جامع النفايات في العادة لن يكون لديه فكرة عن كيفية تحرير تلك المصادر وبالرغم من أنه قد يكون قادرا على التعامل مع بعض هذه المصادر وربما لن يكون من الضروري تفريغهم بشكل دوري. ومن الهام أن تقوم ببعض التنظيف اليدوي وخاصة عندما تتعامل مباشرة مع ما يمثل بعض المصادر على الجهاز.

تجاوز Override الطريقة Object.Finalize وحرر بعض المصادر هناك ففي الحالة الافتراضية هذه الطريقة لا تفعل شيئا ولكن انتبه بما أن جامع النفايات يعمل باستخدام الطريقة Finalize فإن ذلك سيؤثر على الأداء فلا تقم باستخدامها إلا إذا كنت حقا بحاجة لاستخدامها. وحتى عندما يتم تجاوز Override الطريقة Finalize وتحرير المصادر لا يمكنك التحكم بمتى يقوم جامع النفايات ببدء عملية الجمع. مع أنه يمكنك دوما استدعاء System.GC.Collect ولكن لا يجب عليك القيام بذلك في بيئة الإنتاج. فإدارة مصادر CLR عملية معقدة والجامع يعرف متى يكون الوقت الأنسب لبدء دورة الجمع. كما أن أداء تطبيقك سيتأثر بشكل سئ إذا بدأت باستدعاء System.GC.Collect بشكل متكرر وأبق في ذهنك أن جامع النفايات يبدأ عندما يكون هناك حاجة لذاكرة مدارة وليس لديه تحكم حقيقي بالمصادر غير المدارة. حيث توفر لك الـ Framework حلا لهذه المشكلة وهي الواجهة IDisposable

الواجهة IDisposable

إذا كيف يمكنك معرفة إذا كان الشئ Object يحتاج إلى تنظيف إضافي؟ في الحقيقة الجواب سهل جدا فجميع الفئات Classes التي تقوم بتخصيص مصادر يجب تفريغها مباشرة يحب عليها أن تحقق الواجهة IDisposable وهي تحتوي على طريقة وحيدة

```
Public Interface IDisposable
    Sub Dispose()
End Interface
```

إذا كانت الفئة Class التي تستخدمها تحقق هذه الواجهة يجب عليك استدعاء Dispose عندما تنتهي منها. أمر بسيط أليس كذلك؟ ولكن ما هي الفوائد من الواجهة IDisposable؟ الواجهة IDisposable تجعل من الممكن تحديد جميع الفئات Classes التي تخصص مصادر ثمينة وهي توفر آلية مبسطة يمكن الاعتماد عليها لتحرير جميع هذه المصادر. وكما ذكرنا سابقا العديد من الفئات في الـ Framework تحقق الواجهة IDisposable وهذا مثال يستخدم الفئة SqlConnection

```
Dim conn As New SqlConnection()
' Do Stuff
conn.Dispose()
```

ولكن ليس بالضرورة أن يكون هذا موثوقا فحدوث الأخطاء والاستثناءات متوقعا دائما لهذا يجب وضع استدعاء Dispose في قسم Finally في حلقة Catch ... Try مما يضمن أن شيفرتك ستقوم بالتنظيف دائما حتى عند حدوث خطأ أو استثناء ما وبذلك يمكن إعادة كتابة الشيفرة السابقة بالشكل

```
Dim conn As SqlConnection
Try
    conn = New SqlConnection()
    ' Do Stuff
Catch ex as Exception
    ' Handle Exception Here
Finally
    conn.Dispose()
End Try
```

ولكن ماذا إذا احتجنا لتحقيق الواجهة IDisposable؟

تحقيق الواجهة IDisposable

إليك سؤال مثير للاهتمام: متى يجب على الفئة أن تحقق الواجهة IDisposable؟ هذه ثلاث حالات عامة يجب عليك فيها تحقيق الواجهة IDisposable في فئتك:

- عندما تحتوي فئتك على فئات أخرى تحقق الواجهة IDisposable
- عندما تحتوي فئتك على واجهات لأشياء COM
- عندما تحتوي فئتك على مقبض Handle لمصادر Win32 فعالة

وأبسط تحقيق للواجهة IDisposable يمكن أن يكون على الشكل

```
Public Class MyDisposableClass
     Implements IDisposable

 '    Other members

    Public Overloads Sub Dispose() Implements IDisposable.Dispose
         'Release Your Resources Here
    End Sub

End Class
```

ولكن أبسط تحقيق لا يكون دائما الأكثر وثوقية. ويمكن أن يبدو تحقيق أكثر اكتمالا على الشكل

```
Public Class MyDisposableClass
    Implements IDisposable
    Public Overloads Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
        GC.SuppressFinalize(Me)  ' No need call finalizer
    End Sub

    Protected Overridable Overloads Sub Dispose(ByVal disposing As Boolean)
```

```
        If disposing Then
            ' Free managed resources
        End If
        ' Free unmanaged resources
    End Sub

    Protected Overrides Sub Finalize()
        Dispose(False)
    End Sub
End Class
```

بالرغم مما قد يعتقده البعض تبقى إدارة الموارد جزء هام من عملية تطوير التطبيقات باستخدام Visual Basic .net وربما تفترض أنه على جامع النفايات القيام بجميع عمليات التنظيف وإدارة الذاكرة من أجلك بحيث يكون هذا الأمر صحيحا في أغلب الأحيان . وتكمن المشكلة هنا في أن التطبيق يمكن أن يستخدم مصادر معينة لا يمكن لجامع النفايات التعامل معها.مع أن جامع النفايات يمكنه التعامل مع معظم أمور إدارة الذاكرة لتطبيقك إلا أنه في بعض الحالات قد تضطر للقيام بعمل إضافي

و بسبب طبيعة جامع النفايات الغير قابلة للتحديد تظهر مشكلة حيث لا توجد أية ضمانات حول أية أشياء Objects سيتم تنظيفها في النهاية. حيث يشكل هذا الأمر مشكلة عندما تمزج معايير أداء صارمة مع أشياء Objects تستخدم ذاكرة غير مدارة كمقابض الملفات ورسوميات GDI+ ومقابض النوافذ و COM Objects وأشياء أخرى فإذا كان لديك تطبيق أو خدمة تتعامل مع الكثير من الطلبات المشابهة فقد تستنفذ موارد النظام بوقت قصير إن لم تتخلص منهم بسرعة. وبكلمات أخرى قد يسبق تطبيقك عملية التنظيف التي يقوم بها جامع النفايات ويتجاوز المصادر المتاحة من قبل النظام قبل أن يقوم جامع النفايات بتنظيف الأشياء الغير مستعملة.

من الواضح أن هذا الأمر ممكن أن يشكل مشكلة حقيقية ليس فقط بعملية التنظيف بذاتها ولكن بتحديد الأشياء التي تحتاج لتنظيف بشكل فوري. ويكون حل الـ CLR لهذه المشكلة بتقديمه الواجهة IDisopsable حيث تقوم هذه الواجهة بتحديد دالة وحيدة IDisposable.Dispose حيث تكون فكرة الواجهة IDisposable بسيطة جدا فإن كان شيئك Your Object يستخدم مصادر غير مدارة تستمر طوال فترة حياة ذلك الشئ Object يجب عليك تحقيق الواجهة IDisposable وتنطبق هذه القاعدة على فئاتك الخاصة Your Classes وأيضا على الفئات الموجودة سلفا ضمن الـ Framework

كما تجدر ملاحظة أن Visual Basic .net أو الـ CLR يجبرك على القيام بتنظيف الموارد الغير مدارة في فئاتك الخاصة أو تحقيق الواجهة IDisposable وتعتبر هذه هي الممارسة التي ينصح بها وبالتالي تضمن أن المطورين الآخرين سيعلمون أي الفئات تحتاج إلى عملية تنظيف إضافية

Using Generics with Interfaces

Using Generic Interface

يمكن للواجهات Interfaces أن تكون Generics بحيث يمكنها تزويد نوع واحد أو أكثر من المحددات التي تملأ بنوع المحدد عندما يتم استخدام الواجهة انظر للمثال التالي

```
Option Strict Off
Interface IGeneric(Of T)
        Sub SomeMethod(ByVal x As T)
End Interface

Class A
        Implements IGeneric(Of Integer)

        Public Sub SomeMethod(ByVal x As Integer) _
                Implements IGeneric(Of Integer).SomeMethod
                Console.WriteLine("A.SomeMethod received " + x.ToString())
        End Sub
End Class

Class B
        Implements IGeneric(Of Double)

        Public Sub SomeMethod(ByVal x As Double) _
                Implements IGeneric(Of Double).SomeMethod
                Console.WriteLine("B.SomeMethod received " + x.ToString())
        End Sub
End Class

Public Class EntryPoint
        Shared Sub Main()
                Dim ca As IGeneric(Of Integer) = New A()
                Dim cb As IGeneric(Of Double) = New B()
                ca.SomeMethod(123.456)
                cb.SomeMethod(123.456)
        End Sub
End Class
```

فالواجهة IGeneric(Of T) في المثال السابق تستخدم محدد النوع T في كلا الواجهة والطريقة و الفئة Class A تحقق الواجهة IGeneric مستخدما النوع Integer وبالتالي يصبح محدد الوظيفة SomeMethod من النوع Integer وفي الفئة Class B يحققها من النوع Double وبوجود الموجه Option Strict على الوضع Off يسمح بتحويل تضييق ليحول القيمة من النوع Double إلى النوع Integer

Using a Generic Method in an Interface

ليس ضروريا أن تكون الواجهات Generic حتى يكون لها عناصر Generic انظر المثال التالي

```
Option Strict On
Interface INonGeneric
	Sub SomeMethod(Of T)(ByVal x As T)
End Interface

Class A
	Implements INonGeneric

	Public Sub SomeMethod(Of T)(ByVal x As T) _
		Implements INonGeneric.SomeMethod
		Console.WriteLine("A.SomeMethod received " + x.ToString())
	End Sub
End Class

Public Class EntryPoint
	Shared Sub Main()
		Dim ca As INonGeneric = New A()
		ca.SomeMethod(123.456)
		ca.SomeMethod("123 point 456")
	End Sub
End Class
```

الواجهة INonGeneric في المثال السابق ليست Generic ولكن الطريقة SomeMethod(Of T) هي كذلك وعكس المثال السابق فالطريقة ليست محدود بنوع محدد من أجل فئة معينة تحقق تلك الواجهة وهذا المثال يستخدم الموجه Option Strict على الوضع On ليحد من تحويلات التضييق ويثبت أن الأنواع المختلفة مقبول كمحددات

نظرة ضمن مجال الأسماء MY

My Namespace

يزودنا مجال الأسماء My بنية شجرية مفهومة قابلة للفهم سريعا لإجرائيات الفريمورك تستهدف مجموعة من المهام التي تحتاجها بشكل متكرر فعندما نريد قراءة ملف نصي يكون الحل ضمن VB2003 - Framework 1.1 كالتالي

```
Dim sr As New IO.StreamReader("c:\mytextfile.txt")
contents = sr.ReadToEnd
sr.Close()
```

وهذا الكود يعمل جيدا في VB2005 ولكن يمكنك كتابته بشكل أسرع كالكود التالي

```
contents = My.Computer.FileSystem.ReadAllText("c:\mytextfile.txt")
```

الفئة My.Computer.FileSystem هي واحدة من عد فئات ضمن امتداد اللغة My لذا أول شيئ يفترض أن تعمله هو التعرف عليها وما توفره لك فإيجاد وتنفيذ العملية الصحيحة هي عمل روتيني في البرمجة فعندما نريد تنفيذ عملية نستخدمها في عدة أمكنة بشكل متكرر نضمنها ضمن إجراء مساعد لتنفيذ العملية وهذا الإجراء المساعد هو إجراء يأخذ عدة محددات مهمة لك ويقوم بتنفيذ عمل ما بناء عليها مما يسهل عملية البرمجة وهذه عملية يجب على مطوري البرامج القيام بها. ومجال الأسماء My يستخدم هذه الفكرة لتقديم الكثير من الوظائف الهامة للجميع حيث يمكنك اعتبارها مكتبة كاملة لتحسين الإنتاجية حيث يقدم مجال الأسماء My العديد من الإجرائيات مصنفة ضمن العديد من الفئات المختلفة My.Resources, My.Forms, My.Computer, My.Application، My.User, My.Settings, My.WebServices وفي العادة يقدم وصولا سهلا للمعلومات ضمن احد تصنيفات أساسية في الفريموورك My.Application، My.Computer, My.User و المشروع My.Forms, My.Resources، My.Settings,My.WebServices

My.Application

يوفر للمطورين معلومات حول التطبيق والخدمات كمعلومات التطبيق والعنوان والوصف ... الخ فـ OpenForms يوفر معلومات حول النوافذ المفتوحة في المشروع الحالي و Log يوفر إمكانيات للتعامل مع السجلات كتسجيل سجل للأخطاء الحاصلة في البرنامج انظر الكود التالي كمثال

```
Dim winINIFile As String
Try
   winINIFile = My.Computer.FileSystem.ReadAllText("c:\windows\wind.ini")
Catch ex As IO.FileNotFoundException
   My.Application.Log.WriteException(ex, TraceEventType.Error,
       "Error Accessing INI File")
End Try
```

و مجموعة OpenForms أضافت إمكانية شبيهة لما كان موجودا في VB6 حيث توفر مثلا وسيلة سهلة للتنقل عبر جميع النوافذ المفتوحة في التطبيق من دون عناء عمل متغيرات عامة أولا انظر الكود

```
For Each f As Form In My.Application.OpenForms
   Debug.WriteLine(f.Text)
   f.WindowState = FormWindowState.Minimized
Next
```

My.User And My.Computer

My.Compute يمكنك من التنقل عبر الخدمات والمعلومات حول الكمبيوتر المضيف فالفئة FileSystem توفر إجرائيات سهلة للعمل والاستعلام حول الملفات فمثلا الكود التالي يستخدمها لنسخ جميع الصور الموجودة في مجلد My Pictures الخاص بالمستخدم الحالي إلى المجلد C:\Desktop Wallpaper ويعرض مؤشر تقدم إن كانت عملية النسخ تأخذ وقتا أكثر من لحظات قليلة

```
Dim myPics As String = _
   My.Computer.FileSystem.SpecialDirectories.MyPictures

       My.Computer.FileSystem.CopyDirectory( _
           myPics, "C:\Desktop Wallpaper", _
           FileIO.UIOption.AllDialogs, _
           FileIO.UICancelOption.DoNothing)

       MessageBox.Show(My.Computer.FileSystem.GetFiles( _
           myPics, FileIO.SearchOption.SearchAllSubDirectories, _
           "*.jpg", "*.bmp").Count)
```

الفئة My.Computer.Ports تجعل عملية القراءة والكتابة من المدخل التسلسلي عملية بسيطة وهذه تعتبر من العمليات الصعبة ضمن الفريموورك فلعمل نفس وظيفة الكود التالي كان يتطلب كتابة الكثير من الكود سابقا

```
Dim comport As IO.Ports.SerialPort
comport = My.Computer.Ports.OpenSerialPort("COM1")
AddHandler comport.ReceivedEvent, AddressOf DataReceived
```

My.Computer.Audio يمكنك من عزف ملف خاص بالمستخدم أو أحد أصوات النظام

```
Dim musicFile As String
musicFile = My.Computer.FileSystem. _
   GetFiles("C:\WINDOWS\Media", _
   FileIO.SearchOption.SearchAllSubDirectories, _
   "*.wav")(0)
My.Computer.Audio.Play(musicFile)
```

My.Computer.Network يوفر مجالا واسعا من استدعاءات الشبكة كإرسال واستقبال ping من آلة بعيدة أو رفع وتحميل الملفات أو تحديد وضع اتصالك وبعض الأعمال الأخرى

```
If My.Computer.Network.IsAvailable Then
   If My.Computer.Network.Ping("www.duncanmackenzie.net") Then
       Debug.WriteLine("Site Available")

       My.Computer.Network.DownloadFile( _
           "http://www.duncanmackenzie.net/Articles/", _
           System.IO.Path.Combine( _
                 My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
                 "articles.html"))
   End If
End If
```

وهناك العديد من الفئات المثيرة للاهتمام مثل My.Computer.Keyboard، My.Computer.Screen ,My.Computer.Registry يمكنك محاولة استكشافها بنفسك

My.User

من أبسط الفئات ضمن المجال My ومع ذلك فهي توفر الوصول لمعلومات هامة حول المستخدم الحالي للتطبيق

```
If My.User.IsAuthenticated Then
   If My.User.IsInRole("BUILTIN\Administrators") Then
       MsgBox("tsk, tsk... running as Admin are we?")
   End If
End If
```

عرض الإعدادات وأشياء أخرى من تطبيقك

القسم الذي يركز على تطبيقك ضمن المجال My يتضمن Settings, Resources، WebServices, Forms الاثنان الأولان يتعاملان مع البيانات المرتبطة بتطبيقك فمن ضمن بيئة التطوير يمكنك ضبط العديد من عناصر المصادر المختلفة وفي حالتي ضبط الـ Resources أو Settings لمشروعك يمكنك الوصول المباشر إليها عبر My ففور إضافتها لمشروعك توفر الفئات My.Settings , My.Resources تعاملا سهلا مع تلك المعلومات فالكود التالي يقرأ ويعدل قيمة إعداد للمستخدم ثم يستخدم الفئة My.Resources لقراءة نص مخصص

```
Dim lastRun As Date My.Settings.LastRun
My.Settings.LastRun = Now()

Dim myMessage As String = _
   String.Format(My.Resources.LastRunMessage, _
       lastRun.ToShortDateString)

MsgBox(myMessage)
```

بالإضافة إلى ذلك جميع النوافذ في مشروعك موجودة ضمن الفئة My.Forms و أي خدمات ويب موجودة ضمن الفئة My.WebServices وهذا يعني أنه يمكن الوصول إليهم مباشرة فإذا عملت ربطا لخدمة WeatherForecast من webservicex.net مثلا يمكنك قراءة درجة الحرارة كالكود

```
Dim tempService As New net.webservicex.www.WeatherForecast()
Dim wf As net.webservicex.www.WeatherForecasts
wf = tempService.GetWeatherByZipCode("98052")
MsgBox(wf.ToString())
```

ويمكنك تبسيط الأمور كما يلي

```
MsgBox(My.WebServices.WeatherForecast.GetWeatherByZipCode("98052").ToString())
```

My.Forms يختلف عن My.Application.OpenForms بسبب أنه يوفر تواجدا لكل فئة من فئات نماذجك بدون شرط أن تكون مفتوحة حاليا فإذا كنت مبرمج VB6 سابقا ستجد أنها مفيدة كثيرا كالمثال

```
Private Sub showForm2_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles showForm2.Click
   My.Forms.Form2.Show()
End Sub

Private Sub updateForm2_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles updateForm2.Click
   My.Forms.Form2.Text = "Updated..."
End Sub
```

وفي الختام يوفر مجال الأسماء My في VB2005 طريقة سريعة وسهلة للوصول إلى وظائف عميقة ضمن الفريموورك بدون استخدام الفريموورك بشكل مباشر بالإضافة إلى استعادة بعض المفاهيم القديمة

كيف تقوم بإضافة إجراءاتك الخاصة إلى مجال الأسماء My

لإضافة فئة جديدة إلى مجال الأسماء My كل ما عليك عمله هو إضافة كتلة Namespace تحمل اسم My فإذا أردنا إضافة فئة جديدة إلى مجال الأسماء My مثلا TestClass يحتوي على دالة JustForTest سنقوم ببساطة بكتابة الكود التالي

```
Namespace My

   Public Class TestClass

      Public Shared Function JustForTest(ByVal SomeText As String) As String
         Return SomeText & ", " & Now.ToString("dddd")
      End Function

   End Class

End Namespace
```

و أصبح الآن بإمكاننا استخدامه كبقية الفئات الموجودة سابقا في مجال الأسماء My

```
    TextBox2.Text = My.TestClass.JustForTest(TextBox1.Text)
```

ولكن تجدر الملاحظة هنا أن الدالات التي ستستخدمها هنا يجب أن تكون Shared أو أن تقوم بتضمينها في Module بدلا من Class وذلك لأن جميع الدالات والخصائص الموجود في Module تكون Shared دائما.

كما يمكنك اعتماد على الميزة الجديدة وهي الفئات الجزئية Partial Class أيضا إضافة عناصر أخرى لبعض فئات مجال الأسماء My مثل الفئة My.Computer أو الفئة My.Application ولفعل ذلك نقوم بإنشاء فئة جزئية بالاسم الصحيح وإضافة العناصر الجديدة التي نريد حيث يجب عليك تعريف هذه الفئة باستخدام Friend حتى تتطابق مع الفئة الأصلية الموجودة سابقا

```
Namespace My

'    Extend My.Application Class
   Partial Friend Class MyApplication

      Public Function AppTest() As String
         Return "For Test Purposes"
      End Function

   End Class


End Namespace
```

واستخدامه أيضا كبقية الفئات الموجودة سابقا

```
      TextBox3.Text = My.Application.AppTest
```

وبهذا يصبح الكود الكامل لمثالنا هنا كالتالي

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles  Button1.Click

        TextBox2.Text = My.TestClass.JustForTest(TextBox1.Text)
        TextBox3.Text = My.Application.AppTest
    End Sub

End Class

Namespace My

 '    Add New Class
    Public Class TestClass

        Public Shared Function JustForTest(ByVal SomeText As String) As String
            Return SomeText & ",    " & Now.ToString("dddd dd/MM/yyyy")
        End Function

    End Class

 '    Extend My.Application Class
    Partial Friend Class MyApplication

        Public Function AppTest() As String
            Return "For Test Purposes"
        End Function

    End Class

End Namespace
```

ويمكنك الآن توسيع مجال الأسماء My بحسب حاجتك وهذا ليس محدودا بتطبيق معين حيث يمكنك إنشاء مكتبة Class Library خاصة بك تضيف بعض الأشياء لمجال الأسماء My واستخدامها في تطبيقاتك وذلك بإضافة مرجع لهذه المكتبة من داخل التطبيق

كيف تستطيع إطلاق أحداثك الخاصة RaiseEvent Tutorial

أنشئ مشروعا جديدا وسمه RaiseEventsTest ثم أضف له Class واقبل الاسم الافتراضي Class1
حتى نطلق أحداثنا الخاصة ونلتزم بالصيغة التي نشاهدها في الأحداث الخاصة بالتحكمات مثل

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles Button1.Click
```

حيث نجد أن لكل حدث قيمتان دوما هي sender المرسل للحدث و e القيم الممررة بواسطة الحدث

سنقوم أولا باشتقاق فئة من الفئة System.EventArgs ونضمنها القيم التي نريد تمريرها وفائدة هذا الإجراء هو مرونة البرنامج لاحقا عندما تريد التعديل فإذا أردت إضافة قيمة جديدة ليتم تمريرها لن تحتاج سوى لإضافتها هنا و هذا نص الفئة الجديدة

أدخل هذا الكود في الملف الجديد class1

```
Public Class CustomEventArgs
    Inherits System.EventArgs

    Private m_Member1 As String
    Private m_Member2 As Integer

    Public ReadOnly Property Member1() As String
        Get
            Return m_Member1
        End Get
    End Property

    Public ReadOnly Property Member2() As Integer
        Get
            Return m_Member2
        End Get
    End Property

    Public Sub New(ByVal M1 As String, ByVal M2 As Integer)
        m_Member1 = M1
        m_Member2 = M2
    End Sub
End Class
```

حيث عرفنا خاصيتين member1 و member2 لتعيدا القيم التي نريدها و أنشأنا sub new لتهيئة القيم قبل الإرسال الآن بعد تعريف class1أدخل تعريف الحدث الذي نريد إطلاقه كما يلي

```
Public Class Class1

   Public Event TestEvent(ByVal sender As Object, ByVal e As CustomEventArgs)
```

ثم سنضيف إجراء لنطلق الحدث الذي نريده من داخل class1 كالكود التالي حيث نمرر القيم التي نريد إرسالها كمحددات للمشيد constructorالخاص بالفئة customeventargs ثم نستخدم الأمر raiseevent لإطلاق الحدث

```
   Public Sub DoTestEvent()
```

```
        Dim e As New CustomEventArgs("Member No 1", 1500)
        RaiseEvent TestEvent(Me, e)
    End Sub
```

وبهذا نكون قد انتهينا من إنشاء class1 ولم يبق سوى أن نختبره وهذا النص الكامل له

```
Public Class Class1

    Public Event TestEvent(ByVal sender As Object, ByVal e As CustomEventArgs)

    Public Class CustomEventArgs
        Inherits System.EventArgs

        Private m_Member1 As String
        Private m_Member2 As Integer

        Public ReadOnly Property Member1() As String
            Get
                Return m_Member1
            End Get
        End Property

        Public ReadOnly Property Member2() As Integer
            Get
                Return m_Member2
            End Get
        End Property

        Public Sub New(ByVal M1 As String, ByVal M2 As Integer)
            m_Member1 = M1
            m_Member2 = M2
        End Sub

    End Class

    Public Sub DoTestEvent()
        Dim e As New CustomEventArgs("Member No 1", 1500)
        RaiseEvent TestEvent(Me, e)
    End Sub

End Class
```

و لاختبار ما فعلناه اذهب إلى form1 وضع عليها زرا وانقر عليه نقرا مزدوجا فتنتقل لمحرر الكود حيث سنقوم بتعريف متغير يشير إلى class1وذلك خارج إجراء حدث النقر على الزر في منطقة التعريفات العامة في الفئة

```
Private WithEvents cls As New Class1
```

لاحظ استخدام العبارة withevents في تعريف المتغير الذي يشير لـ class1 حتى نستطيع التعامل مع الأحداث التي يطلقها ثم من القائمة المنسدلة اليسارية فوق محرر الكود اختر cls ثم من القائمة المنسدلة اليمينية اختر الحدث testevent فيضيف محرر الكود إجراء التعامل مع الحدث الخاص بنا أدخل الكود ضمن محرر الشيفرة ليصبح الإجراء الجديد كالتالي

```
    Private Sub cls_TestEvent(ByVal sender As Object, _
           ByVal e As Class1.CustomEventArgs) Handles cls.TestEvent
```

```
    MsgBox(e.Member1 & ControlChars.CrLf & e.Member2)
End Sub
```

الآن انتقل إلى حدث النقر على الزر وعدله ليصبح كالتالي

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                 ByVal e As System.EventArgs) Handles Button1.Click

    cls.DoTestEvent()
End Sub
```

الطرائق المُوَسِّعة Extension Methods

يقدم فيجول بايزيك 2008 الطرائق المُوَسِّعة Extension Methods التي تمكن المطور من إضافة وظائف مخصصة لأنواع البيانات المعرفة سابقا من دون إنشاء نوع جديد مشتق. مما يمكنك من كتابة طريقة يمكن أن تستدعى كما لو كانت من ضمن النوع الموجود. والطريقة الموسعة يمكن أن تكون إجراء Sub أو وظيفة Function ولا يمكن أن تكون خاصية Property أو حقل Field أو حدث Event وجميع الطرائق الموسعة يجب تعليمها بالصفة <Extension()> من مجال الأسماء System.Runtime.CompilerServices وتحدد الوسيطة الأولى في الطرائق الموسعة نوع البيانات المراد الذي سيطلق الطريقة.

في المثال التالي يتم تحديد الطريقة Print لتوسيع نوع البيانات string والتي تستخدم Console.WriteLine لإظهار النص حيث يؤسس الوسيط aString أن الطريقة Print توسع الفئة String

```
Imports System.Runtime.CompilerServices

Module StringExtensions

    <Extension()> _
    Public Sub Print(ByVal aString As String)
        Console.WriteLine(aString)
    End Sub

End Module
```

لاحظ أن تعريف الطريقة الموسعة محدد بالصفة <Extension()> ويكون تعليم الـ Module الحاوية للطريقة اختياري ولكن كل طريقة موسعة يجب أن يتم تعليمها بالصفة المذكورة كما يجب استيراد مجال الأسماء System.Runtime.CompilerServices حتى نتمكن من الوصول إلى تلك الصفة. ولا يمكن تعريف الطرائق الموسعة إلا ضمن module ونموذجيا تعرف الطريقة الموسعة في نفس الـ Module التي ستستدعيها وبدلا عن ذلك يتم استيرادها في المكان الذي سنحتاج لاستخدامها فيه فبعد الـ Module التي تحتوي على الطريقة Print يمكن استدعاء الطريقة عندما يكون هناك طريقة لا تأخذ وسائط مثل ToUpper

```
Imports ConsoleApplication2.StringExtensions

Module Module1

    Sub Main()

        Dim example As String = "Hello"
        ' Call to extension method Print.
        example.Print()

        ' Call to instance method ToUpper.
        example.ToUpper()
        example.ToUpper.Print()

    End Sub

End Module
```

وفي المثال التالي تكون PrintAndPunctuate طريقة موسعة أخرى للنوع String وفي هذه المرة تمتلك وسيطتين الأولى aString تحدد أن الطريقة توسع النوع String وتكون الوسيطة الثانية punc عبارة عن سلسلة من علامات الترقيم التي ستمرر عند استدعاء الطريقة التي تظهر نصا متبوعا بعلامات الترقيم

```
<Extension()> _
Public Sub PrintAndPunctuate(ByVal aString As String, _
                             ByVal punc As String)
    Console.WriteLine(aString & punc)
End Sub
```

حيث يتم استدعاء الطريقة بتمرير وسيطة نصية لـ punc مثلا `example.PrintAndPunctuate(".")`

ويظهر المثال التالي تحديد واستدعاء Print و PrintAndPunctuate واستيراد System.Runtime.CompilerServices الذي يمكن من الوصول إلى الصفة التي تحدد الطرائق الموسعة

```
Imports System.Runtime.CompilerServices

Module StringExtensions

    <Extension()> _
    Public Sub Print(ByVal aString As String)
        Console.WriteLine(aString)
    End Sub

    <Extension()> _
    Public Sub PrintAndPunctuate(ByVal aString As String, _
                                 ByVal punc As String)
        Console.WriteLine(aString & punc)
    End Sub

End Module
```

ثم يتم استدعاء الطرائق الموسعة

```
Imports ConsoleApplication2.StringExtensions
Module Module1

    Sub Main()

        Dim example As String = "Example string"
        example.Print()

        example = "Hello"
        example.PrintAndPunctuate(".")
        example.PrintAndPunctuate("!!!!")

    End Sub
End Module
```

كل ما يتطلبه تشغيل طرائق موسعة كهذه هي أن تكون ضمن مجال رؤية الكود فإن كانت Module ضمن المجال فستكون الطريقة مرئية من قبل IntelliSense ويمكن استدعاؤها كما لو كانت من ضمن الطرائق الاعتيادية.

لاحظ انه عندما يتم استدعاء الطرائق الموسعة لا يتم تمرير قيم للوسيطة الأولى فالوسيطة aString في الطريقة السابقة تحدد أنها يجب أن تستدعى من النوع string والمترجم سيستخدم المتغير النصي كقيمة لتلك الوسيطة

ويمكن تحديد طرائق موسعة على معظم الأنواع التي يمكن استخدامها في فيجول بايزيك ضمن قائمة الوسائط متضمنا التالي:

- الفئات (الأنواع المرجعية) Classes (Reference Types)
- التراكيب Structures
- الواجهات Interfaces
- الإجراءات المفوضة Delegates
- وسائط ByRef و ByVal
- وسائط الأنواع Generic
- المصفوفات Arrays

توفر الطرق الموسعة طريقة مريحة وقوية لتوسيع نوع موجود ومع ذلك كي يمكن استخدامهم بنجاح هناك بعض النقاط التي يجب أخذها بعين الاعتبار وهي تنطبق بشكل رئيس على مطوري مكتبات الفئات Class Libraries ولكنها يمكن أن تؤثر على أي تطبيق يستخدم الطرائق الموسعة. وبشكل عام فالطرائق الموسعة التي تضيفها للأنواع التي لا تملكها تكون أضعف من تلك التي تملك التحكم عليها لوجود عدد من الأشياء التي قد تحدث عندما تضاف طرائق موسعة للأنواع التي لا تملكها فيمكن أن تتداخل تلك الطرائق مع عمل طريقتك الموسعة

- عند وجود عنصر عضو في الفئة يملك توقيعا متوافقا مع وسائط التعبير المستدعي بدون حدوث تحويل تضييق narrowing Conversion مطلوب من الوسائط فيملك ذلك العنصر الأفضلية على طريقتك الموسعة لذلك يمكن عند إضافة بعض الخصائص للفئة أن لا يعود بالإمكان الوصول لطريقتك الموسعة
- لا يمكن أن يمنع كاتب الطريقة الموسعة المبرمجين الآخرين من كتابة طرائق موسعة تتعارض مع طريقته
- يمكنك تحسين قوة إجراءاتك الموسعة بوضعها ضمن مجال أسماء namespace خاص بهم لتمكين مستخدمي مكتبتك من استيراد أو استبعاد كامل مجال الأسماء أو اختيار المناسب من مجالات أسماء المكتبة
- يكون توسيع الواجهات Interfaces آمن من توسيع الفئات Classes وخاصة إذا كنت لا تملك تلك الواجهة أو الفئة فكل تغيير في واجهة سيؤثر على جميع الفئات التي تعتمد عليها لذلك فالمبرمج قليلا ما يغير في الطرائق الموجودة في الواجهة
- قم بتوسيع نوعا محددا قدر الإمكان فعندما توسع نوعا تشتق منه أنواع أخرى تكون هناك طبقات من الاحتمالات من الطرائق العادية والموسعة التي قد تتداخل مع طريقتك

عندما يتطابق توقيع طريقة عادية Instance Method مع طريقة موسعة Extension Method في نفس المجال لا يمكن الوصول للطريقة الموسعة بالنتيجة. لهذا لا يمكن أن نقوم باستخدام طريقة موسعة لاستبدال طريقة عادية ومع ذلك يمكن للطرائق الموسعة امتلاك نفس الاسم مع الطرائق العادية ولكن مع توقيع مختلف فتكون كلتا الطريقتان متوفرتان طالما أن التوقيع مختلف فمثلا إن كانت الفئة ExampleClass تحتوي على طريقة exampleMethod لا تأخذ أية وسائط يمكن لطريقة موسعة تمتلك نفس الاسم أن تتواجد ولكن بتوقيع مختلف

```
Imports ConsoleApplication2.ExtensionExample
Module Module1

    Sub Main()
        Dim ex As New ExampleClass
        ' The following statement calls the extension method.
        ex.exampleMethod("Extension method")
        ' The following statement calls the instance method.
        ex.exampleMethod()
    End Sub

    Class ExampleClass
        ' Define an instance method named exampleMethod.
        Public Sub exampleMethod()
            Console.WriteLine("Instance method")
        End Sub
    End Class

End Module
Imports System.Runtime.CompilerServices

' Define an extension method named exampleMethod.
Module ExtensionExample
    <Extension()> _
    Sub exampleMethod(ByVal ec As ExampleClass, _
                      ByVal stringParameter As String)
        Console.WriteLine(stringParameter)
    End Sub

End Module
```

والخرج الناتج عن الكود السابق يكون كما يلي

```
Extension method

Instance method
```

عندما يكون لطريقتين موسعتين توقيعان مطابقان في نفس مجال الوصول يتم استدعاء الطريقة التي تملك الأسبقية العليا حيث يتم تحديد هذه الأسبقية عبر طريقة إدخال هذه الطريقة ضمن المجال وتمثل القائمة التالية تسلسل الأسبقية التالي:

١. الطرائق الموسعة الموجودة سابقا ضمن نفس الـ Module
٢. الطرائق الموسعة المعرفة ضمن أنواع البيانات في مجال الأسماء الحالي أو أحد آبائه حيث تملك مجالات الأسماء الأبناء أسبقية على الآباء
٣. الطرائق الموسعة المعرفة ضمن أي نوع تم استيراده للملف الحالي
٤. الطرائق الموسعة الموجودة ضمن أي مجال أسماء مستورد ضمن الملف الحالي
٥. الطرائق الموسعة المعرفة ضمن أي نوع مستورد على مستوى المشروع
٦. الطرائق الموسعة المعرفة ضمن أي مجال أسماء مستورد على مستوى المشروع

فإن لم تحل تلك الأسبقية المشكلة يمكنك عندها استخدام الاسم الكامل للطريقة الموسعة لتحديد الطريقة التي تقوم باستدعائها فإن كانت الطريقة Print في المثال السابق محددة ضمن الـ Module المسماة StringExtensions يكون الاسم الكامل للطريقة الموسعة هو StringExtensions.Print(example) بدلا عن .example.Print()

الطريقةMain

النوع Module

يدعم فيجول ستوديو وحدة بناء برمجية تدعى Module والتي يتم التصريح عنها باستخدام الكلمة المحجوزة Module فعندما تقوم بإنشاء تطبيق كونسول مثلا يتم إنشاء ملف vb من أجلك يحتوي على الكود التالي:

```
Module Module1
    Sub Main()

    End Sub
End Module
```

وفي الواقع فإن الـ Module هي عبارة عن فئة Class مع بعض الاختلافات الواضحة فأي إجراء أو وظيفة عامة Public يتم تعريفها ضمن الـ Module تكون عنصر مشترك Shared Member يمكن الوصول إليه مباشرة من جميع أنحاء التطبيق وبذلك فأنت لست مضطرا لكتابة اسم الـ Module عندما تريد استخدام عناصرها

```
Module Module1
     Sub Main()
          ' Show banner.
          DisplayBanner()
          ' Get user's name and say howdy.
          GreetUser()
     End Sub

     Sub DisplayBanner()
          ' Get the current color of the console text.
          Dim currColor As ConsoleColor = Console.ForegroundColor
          ' Set text color to yellow.
          Console.ForegroundColor = ConsoleColor.Yellow
          Console.WriteLine("******* Welcome to FunWithModules *******")
          Console.WriteLine("This simple program illustrates the role")
          Console.WriteLine("of the Module type.")
          Console.WriteLine("*****************************************")
          ' Reset to previous color of your console text.
          Console.ForegroundColor = currColor
          Console.WriteLine()
     End Sub

     Sub GreetUser()
          Dim userName As String
          Console.Write("Please enter your name: ")
          userName = Console.ReadLine()
          Console.WriteLine("Hello there {0}. Nice to meet ya.", userName)
     End Sub
End Module
```

لاحظ في المثال السابق أن الطريقة Main يمكنها استدعاء الطريقة GreetUser لأنها معرفة ضمن نفس الـ Module ولست مضطرا لكتابة اسم الـ Module كبادئة عند استدعاء عنصر فيها. وفي التطبيق الذي يحتوي على أكثر من Module أنت لست مضطرا لكتابة اسم أي Moduleكبادئة عند استدعاء العناصر المحتواة في تلك الـ Modules إلا إذا كانت هناك عدة طرائق في أكثر من Module تحمل نفس الاسم.

كما أن الـ Module لا يمكن أن يتم استخدامها كنوع عند التصريح عن متغير باستخدام الكلمة المحجوزة New لأنها غير قابلة للإنشاء Non Creatable وإن استخدمت كودا شبيها بالكود التالي فإن المترجم سيعطيك خطأ

```
' Nope! Error, can't allocated modules!
Dim m as New Module1()
```

كما يمكن لـ Module أن تحتوي على عناصر أخرى إضافة للوظائف والإجراءات كحقول البيانات مثلا كما في المثال

```
Module MyModule
    Public UserName As String

    Sub GreetUser()
        Console.WriteLine("Hello, {0}.", UserName)
    End Sub

End Module
```

وكأي عنصر آخر ضمن Module يمكننا استخدام الحقل UserName مباشرة

```
Sub Main()
    ...
    ' Set user's name and call second form of GreetUser().
    UserName = "Fred"
    MyModule.GreetUser()
    ...
End Sub
```

الطريقة Main

أي تطبيق Vb2008 يجب أن يحتوي على نوع يحدد الطريقة Main التي تشكل نقطة الدخول للتطبيق وهي تكون موجودة عادة ضمن Moduleوهي طريقة مشتركة دوما Shared method مع انه يمكن أن تتواجد ضمن فئة Class عندها يجب عليك التصريح عنها باستخدام الكلمة المحجوزة .Shared وكمثال على ذلك أنشئ تطبيق كونسول جديد واستبدل كامل الكود المولد بالكود التالي

```
Class Program

    ' Unlike Modules, members in a class are not
    ' automatically shared. Thus, we must declare Main()
    ' with the Shared keyword.
    Shared Sub Main()

    End Sub

End Class
```

فإن حاولت تشغيل البرنامج عند هذه النقطة فسوف تتلقى خطأ بأنه لا يمكن إيجاد الطريقة Main ولحل هذه المشكلة يجب عليك تحديد الفئة Program كعنصر بدء للتطبيق وذلك في Startup Object من الصفحة Application في خصائص MyProject

معالجة محددات سطر الأوامر

إحدى المهام الرئيسية للطريقة Main هي معالجة محددات سطر الأوامر التي يتم تمريرها للتطبيق فمثلا مترجم سطر الأوامر vbc.exe يمتلك العديد من الخيارات مثل target و out وغيرها فعندما تريد إنشاء طريقة Main تقوم بمعالجة محددات سطر الأوامر الممررة للتطبيق ستجد أنه لديك عدة طرق للقيام بذلك. وأول طريقة يمكننا استخدامها لهذا الغرض هو استخدام الطريقة GetCommandLineArgs المعرفة ضمن المجال System.Envitronment التي تعيد مصفوفة من العناصر من النوع String وأول عنصر في هذه المصفوفة هو اسم الملف التنفيذي للتطبيق بينما تشكل باقي عناصر المصفوفة محددات سطر الأوامر الممررة للتطبيق

```
Class Program

    Shared Sub Main()
```

```
        Console.WriteLine("***** Fun with Main() *****")
        ' Get command-line args.
        Dim args As String() = Environment.GetCommandLineArgs()
        Dim s As String

        For Each s In args
            Console.WriteLine("Arg: {0}", s)
        Next
    End Sub

End Class
```

وإن لم ترغب باستخدام الطريقة السابقة يمكنك تحديد مصفوفة String كدخل للطريقة Main كما في الكود

```
Shared Sub Main(ByVal args As String())
    Console.WriteLine("***** Fun with Main() *****")
    ' Get command-line args.
    Dim s As String
    For Each s In args
        Console.WriteLine("Arg: {0}", s)
    Next
End Sub
```

استخدام Main كوظيفة وليست إجراء

يمكننا أيضا تعريف Main كوظيفة تعيد قيمة من النوع Integer الأمر الذي يعد من الموروثات من لغة C حيث يعيد البرنامج القيمة 0 عند انتهائه بصورة طبيعية أو يمكن إعادة أي قيمة أخرى تمثل خرجا ما أو رقما للخطأ الذي أدى إلى انتهاء البرنامج كما في الكود

```
Shared Function Main(ByVal args As String()) As Integer
    Console.WriteLine("***** Fun with Main() *****")
    Dim s As String
    For Each s In args
        Console.WriteLine("Arg: {0}", s)
    Next
    ' Return a value to the OS.
    Return 0
End Function
```

وبغض النظر عن الطريقة التي نقوم فيها بتعريف Main كإجراء أو وظيفة يبقى الهدف الأساسي منها هو التعامل مع الأنواع التي تشكل وظيفية البرنامج وتذكر دوما أنه عند انتهاء Main أو الخروج منها فإن البرنامج سيتم إنهاؤه حتما بالنتيجة

تمرير محددات سطر الأوامر من داخل بيئة التطوير

يمكننا فيجول ستوديو من تمرير محددات سطر الأوامر للتطبيق عند اختباره من داخل بيئة التطوير الأمر الذي يسهل علينا عملية الاختبار لما يوفر علينا من عناء تمرير تلك المحددات يدويا حيث يمكننا تمريرها أثناء الاختبار بإدخالها في مربع Command Line Arguments في الصفحة Debug من خصائص MyProject

التحميل الزائد للمعاملات Operators Overloading

يعرف فيجول بايزيك معاملات للتعابير التي تستخدم الأنواع القياسية للبيانات مثل المنطقية أو الحقيقية. وهي تحدد بعض المعاملات الخاصة بالأغراض Objects مثل Is أو IsNot ولكن بعض المعاملات مثل * و Mod لا تعني شيئا بالنسبة لتلك الأغراض بشكل عام.

ومع ذلك يمكنك تعريف معاملات Operators بالنسبة للتراكيب Structures والفئات Classes بحيث تكون الصيغة العامة للمعامل Operator كما يلي

```
[ <attributes> ] Public [ Overloads ] Shared [ Shadows ] _
[ Widening | Narrowing ] Operator symbol ( operands ) As type
...
End Operator
```

وتكون أقسام هذه الصيغة العامة كما يلي:

- Attributes خصائص المعامل
- Public و Shared جميع المعاملات يجب تعريفها باستخدام محدد الوصول Public أو Shared
- Overloads نستخدمه عندما يكون لدينا معامل يأخذ محددين من فئة أساسية وفئة مشتقة وفي هذه الحالة يعني أنك تتجاوز المعامل الموجود في الفئة الأساسية
- Shadows المعامل يستبدل معامل مماثل موجود في الفئة الأساسية
- Widening يبين أن المعامل يحدد تحويل تعريض ينجح دوما في زمن التشغيل وبالتالي فهذا المعامل يجب عليه التقاط مؤشر لجميع الأخطاء الممكن حدوثها. والمعامل CType يجب أن يحتوي على أحد الكلمتين الأساسيتين Widening أو Narrowing
- Narrowing يبين أن المعامل يحدد تحويل تضييق ربما يفشل في زمن التشغيل. والمعامل CType يجب أن يحتوي على أحد الكلمتين الأساسيتين Widening أو Narrowing
- Symbol رمز المعامل ويجب أن يكون أحد القيم التالية: +، -، *، /، \، ^، &، >>، <<، =، <>، >، <، =>، =<، Mod، Not, And, Or, Xor, Like, IsTrue, IsFalse, CType
- Operands تعريف الأغراض التي تتم معالجتها من قبل المعامل بحيث تأخذ المعاملات +، -، Not, IsTrue, and IsFalse محدد واحد والمعاملات +، -، *، /، \، ^، &، >>، <<، =، <>، >، <، =>، =<، Mod, And, Or, Xor, Like, CType تأخذ محددين
- Type جميع المعاملات يجب أن يكون لها نوع بيانات معاد

والتحميل الزائد للمعاملات يكون محددا بعدة ضوابط

- بعض المعاملات تأتي على شكل أزواج وإن قمت بتعريف واحد منها يجب عليك تعريف الآخر وهي:
    - = و <>
    - < و >
    - =< و =>
    - IsTrue و IsFalse
- في المعاملات العادية والمنطقية يجب أن تظهر الفئة المعرفة للمعامل في المعامل. ومن أجل معاملات تحويل النوع يجب على الفئة أو التركيب الظهور في المعامل أو القيمة المعادة
- المعاملان IsTrue و IsFalse يجب أن يعيدا قيمة منطقية
- المعامل الثاني لـ >> و << يجب أن يكون عدد صحيح

إذا عرفت معاملا يستطيع فيجول بايزيك معالجة ذلك المعامل متبوعا بإشارة = بشكل تلقائي فإن عرفت المعامل + فإن فيجول بايزيك يفهم المعامل =+ تلقائيا

طالما أنه لا يمكنك استخدام المعاملين IsTrue و IsFalse بشكل مباشر فيمكنك استخدامهما بشكل غير مباشر فإن قمت بتعريف المعامل IsTrue لفئة ما يستخدمها فيجول بايزيك لتحديد فيما إذا كان استخدام ذلك الغرض كقيمة True في التعبير المنطقي. فعلى سبيل المثال تستخدم العبارة التالية لتحديد فيما إذا كان الغرض c1 يمكن اعتباره True

```
If c1 Then …
```

وإن عرفت المعاملان And و IsFalse فيقوم فيجول بايزيك باستخدامهما لمعالجة المعامل AndAlso أيضا ومن أجل أن يعمل هذا يجب على المعامل And إعادة نفس نوع الفئة أو التركيب الذي عرفته ضمنها. فإن افترضنا أنك عرفت And و IsFalse في الفئة Composite والمتغيرات C1 و C2 و C3 هي تواجدات Instances لتلك الفئة انظر العبارة

```
C3 = C1 AndAlso C2
```

فهنا يستخدم فيجول بايزيك المعامل IsFalse لتقييم C1 فإن أعاد المعامل IsFalse القيمة True فلن يزعج البرنامج نفسه بتقييم قيمة C2 ويفترض أن قيمة العبارة كاملة هي False ويعيد القيمة False بالنسبة للتعبير بكامله وذلك لأن IsFalse أعادت القيمة True بالنسبة لـ C1 ويعرف فيجول بايزيك تلقائيا أنه إذا كانت قيمة C1 هي False فيجعل قيمة C3 مساوية لقيمة C1 وهذا يسبب بعض التشويش ومما يجعل الأمر منطقيا أكثر هو التفكير بأن فيجول بايزيك يقيم التعابير المنطقية التي تستخدم المعامل AndAlso العادي. وبشكل مشابه يمكن تعريف المعاملين Or و IsTrue فيقوم فيجول بايزيك تلقائيا بتزويدنا بالمعامل OrElse

طالما أنك بشكل عام لا يمكنك عمل إجراءين في فيجول بايزيك يختلفان فقط بالقيمة المعادة ولكن يمكن عمل ذلك فقط بالنسبة لمعاملات CType الخاصة بالتحويل بين الأنواع فعندما يحاول البرنامج التحويل بين الأنواع يمكنه تحديد أي معامل يجب عليه استخدامه بمعرفته لنوع القيمة المعادة

وفي الكود التالي نرى الفئة Complex التي تمثل ر قم معقد وتحدد المعاملات + و – و * لتعريف عمليات الجمع والطرح والضرب للأرقام المعقدة ويحدد أيضا المعاملات = و <> ومعاملات التحويل التي تحول الرقم المعقد لقيمة معادة من النوع Double لإعطائها أهميتها

```
Public Class Complex
    Public Re As Double
    Public Im As Double

    ' Constructors.
    Public Sub New()

    End Sub

    Public Sub New(ByVal real_part As Double, ByVal imaginary_part As Double)
        Re = real_part
        Im = imaginary_part
    End Sub

    ' ToString.
    Public Overrides Function ToString() As String
        Return Re.ToString & " + " & Im.ToString & "i"
    End Function

    ' Operators.
```

```
    Public Shared Operator *(ByVal c1 As Complex, ByVal c2 As Complex) As Complex
        Return New Complex( _
        c1.Re * c2.Re - c1.Im * c2.Im, _
        c1.Re * c2.Im + c1.Im * c2.Re)
    End Operator

    Public Shared Operator +(ByVal c1 As Complex, ByVal c2 As Complex) As Complex
        Return New Complex( _
        c1.Re + c2.Re, _
        c1.Im + c2.Im)
    End Operator

    Public Shared Operator -(ByVal c1 As Complex, ByVal c2 As Complex) As Complex
        Return New Complex(c1.Re - c2.Re, c1.Im - c2.Im)
    End Operator

    Public Shared Operator =(ByVal c1 As Complex, ByVal c2 As Complex) As Boolean
        Return (c1.Re = c2.Re) AndAlso (c1.Im = c2.Im)
    End Operator

    Public Shared Operator <>(ByVal c1 As Complex, ByVal c2 As Complex) As Boolean
        Return (c1.Re <> c2.Re) OrElse (c1.Im <> c2.Im)
    End Operator

    Public Shared Operator -(ByVal c1 As Complex) As Complex
        Return New Complex(c1.Im, c1.Re)
    End Operator

    Public Shared Narrowing Operator CType(ByVal c1 As Complex) As Double
        Return System.Math.Sqrt(c1.Re * c1.Re + c1.Im * c1.Im)
    End Operator

End Class
```

إنشاء مكتبة تضيف وظائف جديدة للتحكمات الموجودة بدون استخدام الوراثة

أنشئ مشروعا جديدا من النوع Windows Forms Application كي نقوم بالعمل عليه ثم من قائمة File ومن القائمة الفرعية ADD أضف مشروعا جديدا للمشروع الحالي من النوع Class Library وقم بتسميته TestExtendingLib وفي ملف الكود الخاص بالمكتبة أضف في بدايته المراجع التالية الضرورية لعمل مثالنا هنا بعد التأكد من أنك قد أضفت المراجع المناسبة لها من صفحة References من خصائص My Project

```
Imports System.Runtime.CompilerServices
Imports System.Math
Imports System.Drawing
Imports System.Windows.Forms
```

استبدل التعريف

```
Public Class Class1


End Class
```

بالتعريف

```
Public Module TestExtending

End Module
```

انتهت عملية التهيئة الآن وأصبحنا جاهزين لإضافة توسعاتنا

سأبدأ1 بإضافة الوظيفة Alert لصندوق النصوص التي ستحول لون الخلفية له إلى الأحمر والكتابة إلى الأصفر عند استدعائها - داخل تعريف الـ Module السابق أدخل الكود التالي

```
<Extension()> _
Sub Alert(ByVal TxtBx As TextBox)
    TxtBx.Font = New Font(TxtBx.Font, FontStyle.Bold)
    TxtBx.ForeColor = Color.Yellow
    TxtBx.BackColor = Color.Red
End Sub
```

وسنضيف وظيفة أخرى لتوسيع النوع String بحيث تحول القيمة الموجودة في السلسلة النصية إلى Double ثم تضربها بالعدد الممرر وتعيد الناتج

```
<Extension()> _
Function Multiply(ByVal InString As String, _
                  ByVal Multply As Double) As Double

    Return CDbl(InString) * Multply
End Function
```

قم بعمل Build للمكتبة TestExtending
من أجل التجربة انقر بزر الفأرة اليميني على تطبيق ويندوز الذي أنشأناه في البداية ثم اضغط Add Reference وأضف مرجعا للمكتبة التي أنشأناه للتو ثم أضف الاستيراد التالي في بداية كود النموذج

```
Imports TestExtendingLib.TestExtending
```

أضف TextBox و Button و ListBox للنموذج ثم انقر نقرا مزدوجا على الزر لإنشاء حدث النقر عليه ثم أدخل الكود التالي لاختبار الوظيفتين اللتان أضفناهما سابقا

```
Me.TextBox1.Text = "123.456"
Me.TextBox1.Alert()
```

```
Dim s As String = Me.TextBox1.Text
MsgBox(s.Multiply(13))
```

شغل البرنامج واضغط على الزر وانظر النتيجة

كما ترى أضفنا لصندوق النصوص وظيفة جديدة لم تكن موجودة سابقا تسمى Alert يمكننا الاستفادة منها وتطبيقها على أي صندوق نصوص في أي مشروع نريده فقط بإضافة المكتبة والاستيراد المناسب كما رأينا سابقا كما أضفنا أيضا الوظيفة Multiply لنوع البيانات String ويمكن استخدامه أيضا في أي مشروع بنفس الطريقة

يمكننا أيضا إضافة وظيفة لـ ListBox لتفريغها وملئها ببعض البيانات مثلا بنفس الطريقة حيث يمكنك إضافة الكود التالي للمكتبة ثم عمل ReBuild لها وتجربة كيف أن صندوق القائمة أصبح يمتلك الوظيفة الجديدة

```
<Extension()> _
Sub ClearAndFillComputerCorp(ByVal LstBx As ListBox)
    LstBx.Items.Clear()

    LstBx.Items.Add("Micrsoft")
    LstBx.Items.Add("Sun")
    LstBx.Items.Add("Intel")
    LstBx.Items.Add("IBM")
    LstBx.Items.Add("Borland")
    LstBx.Items.Add("CyberLink")
    LstBx.Items.Add("Nvidia")
    LstBx.Items.Add("Gigabyte")
    LstBx.Items.Add("MSI")
End Sub
```

كما يمكننا أيضا إضافة وظيفة للتحكم Button تغير من مظهره بنفس الطريقة

```
<Extension()> _
Sub YellowOnDarkBlue(ByVal Btn As Button)
    Btn.Font = New Font(Btn.Font, FontStyle.Bold)
    Btn.ForeColor = Color.Yellow
    Btn.BackColor = Color.DarkBlue
End Sub
```

الآن أي وظائف متكررة تستخدمها يمكنك إضافتها للتحكمات أو الفئات الموجودة وأصبحت وكأنه وظائف أساسية وذلك بإنشاء المكتبة الخاصة بك واستيرادها للمشروع

في ملف الكود الخاص بالمكتبة TestExtendingLib فقط قم باستبدال السطر التالي في بداية الملف

```
Public Module TestExtending
```

بالسطر

```
<Global.Microsoft.VisualBasic.HideModuleName()> _
Public Module TestExtending
```

وبذلك عند استخدام المكتبة يمكنك أن تستبدل سطر الاستيراد التالي

```
Imports TestExtendingLib.TestExtending
```

بسطر الاستيراد التالي

```
Imports TestExtendingLib
```

وشرح التعديل هو بما أن كودنا في المكتبة ضمن Module ويمكننا استخدام الدالات الموجودة في الـ Module باسمها فقط لذا أضفنا الواصفة HideModuleName التي تسبب إخفاء اسم الـ Module وبالتالي اختصار الاستيراد عند استدعاء المكتبة

طبعا بعد التعديل لا تنس عمل Rebuild للمكتبة حتى تسري التعديلات الجديدة

توسيع مجال الأسماء My باستخدام My Extensibility

تم تقديم مجال الأسماء My بدءا من Visual Basic 2005 ليوفر اختصارات للطرائق واستدعاءات API الشائعة ومنذ ذلك الوقت كان يمكن للمستخدمين كتابة توسعاتهم الخاصة لمجال الأسماء My مضيفين له مكتبات الكود الخاصة بهم. وقد قدم My Extensiility كجديد في Visual Basic 2008 مما سهل توسيع المجال My. ومع ميزة My Extensibility الجديدة أصبح بالإمكان تفعيل وإلغاء تفعيل My Extensibility من خلال مصمم المشاريع أو عندما يتم ربط أو إزالة مرجع لمجمع في المشروع الأمر الذي جعل عملية توسيع بيئة تطوير Visual studio أكثر بساطة.

طرق توسيع المجال My

هناك القليل من نماذج الكود التي يمكن استخدامها لتوسيع المجال My. والطريقة الأسهل لعمل ذلك هي تلك التي نعرفها سابقا فعملية إضافة أي شئ للمجال My هي عملية مماثلة لعملية إضافة أي شئ لأي مجال أسماء آخر كما في المثال

```
Namespace My.HandyStuffForMy
   <Global.Microsoft.VisualBasic.HideModuleName()> _
   Friend Module HandyStuffForMyModule
      Sub Foo()
         ...
      End Sub

      Property Bar()
         ...
      End Property
   End Module
End Namespace
```

الذي يجعل الإجراء Foo و الخاصية Bar يظهران ضمن المجال HandyStuffForMy ضمن المجال My. حيث استخدمنا الواصفة attribute المسماة HideModuleName التي تمنع اسم الـ module من الظهور لأنك لا تحتاجه للوصول إلى العناصر المحتواة ضمنه. وإن أردت إضافة هذه العناصر للمجال My بدون إنشاء مجال فرعي عدل اسم المجال في الكود السابق ليصبح My فقط كما يلي

```
Namespace My
   <Global.Microsoft.VisualBasic.HideModuleName()> _
   Friend Module HandyStuffForMyModule
      Sub Foo()
         ...
      End Sub

      Property Bar()
         ...
      End Property
   End Module
End Namespace
```

وإن استخدمت اسم مجال موجود سلفا ضمن المجال My كـ Resources مثلا فهذه العناصر سيتم إضافتها لذلك المجال مما يعطيك مرونة كبيرة في كيفية الوصول إلى توسعاتك. والشئ الذي يجب الانتباه إليه هو محدد الوصول فمحدد الوصول Friend سيقوم بمنع المجمعات الأخرى من الوصول إلى هذه العناصر وإن استخدمت محدد الوصول Public ربما تتضارب هذه العناصر مع رموز معرفة في المشاريع الأخرى التي تستخدم المجال My

التوسيع باستخدام الورقة المفردة Singleton

My.Computer و My.Settings و My.Application هي تواجدات وحيدة الورقة Singleton Instances للفئات My.MyComputer و My.MySettings و My.MyApplication حيث يمكننا إضافة عناصر لهذه الفئات بسهولة باستخدام ميزة الفئات المجزأة Partial Class فمثلا الكود التالي يضيف إجراء يدعى Shutdown للغرض My.Computer

```
Namespace My
   Partial Class MyComputer
      Sub Shutdown()
         ...
      End Sub
   End Class
End Namespace
```

كما يمكن استخدام هذا الكود لتوسيع My.Application و My.Settings أيضا حيث تمكنك ميزة الفئات المجزأة من إضافة عناصر إلى فئات معرفة في مكان آخر وهذا يتضمن فئات مثل MyComputer في مشاريع Visual Basic حيث يقوم المترجم بحقن الفئات قبل أن يتم ترجمة الكود.

وإذا أردت جعل عناصرك التي قمت بإضافتها يمكن الوصول إليها كورقة مفردة مثل الفئات المذكورة يجب عليك إضافة خاصية للقراءة فقط للمجال My تعيد تواجدا عاما يماثل ذلك المزود من ThreadSafeObjectProvider(Of T) الذي يقوم بإنشاء صورة من تلك الفئة على مستوى المسلك Thread والكود التالي يرينا كيفية إضافة My.CustomMyObject مع عناصره

```
Namespace My
  <Global.System.ComponentModel.EditorBrowsable _
  (Global.System.ComponentModel.EditorBrowsableState.Never)> _
  Partial Friend Class MyCustomMyObject
    Public Sub Foo()
        ...
    End Sub

    Public Property Bar()
        ...
      End Property
  End Class

  <Global.Microsoft.VisualBasic.HideModuleName()> _
  Friend Module CustomMyObjectModule
    Private instance As ThreadSafeObjectProvider(Of MyCustomMyObject)

    ReadOnly Property CustomMyObject() As MyCustomMyObject
      Get
        Return instance.GetInstance()
      End Get
    End Property
  End Module
End Namespace
```

كتابة التوسع

سنبدأ بتشغيل Visual Studio وإنشاء تطبيق كونسول جديد وتسميته MyValidation وأضافة Module جديدة للمشروع وتسميتها MyValidation.vb ثم استبدل كامل الكود الموجود بالكود التالي

```
Imports System.Text.RegularExpressions
Imports System
Imports System.Linq
Imports Microsoft.VisualBasic

Namespace My.Validation
  <Global.Microsoft.VisualBasic.HideModuleName()> _
  Public Module MyValidationMod
    Public Function IsEmpty(ByVal value As Object) As Boolean
      If (value Is Nothing) OrElse _
         (value Is System.DBNull.Value) OrElse _
         (value.ToString = String.Empty) OrElse _
         (TypeOf value Is Date AndAlso CDate(value) = Date.MinValue) _
      Then
        Return True
      Else
        Return False
      End If
    End Function

    Public Function IsAlphaNumeric(ByVal value As String, _
     Optional ByVal emptyOK As Boolean = False, Optional ByVal _
     whitespaceOK As Boolean = False) As Boolean
      If IsEmpty(value) Then Return emptyOK

      Dim expr As String
      If whitespaceOK Then
        expr = "^[a-zA-Z0-9\s]+$"
      Else
        expr = "^[a-zA-Z0-9]+$"
      End If

      Return Regex.IsMatch(value, expr)
    End Function

    Public Function IsCanadianProvince(ByVal st As String) As Boolean
      Dim allProv = "|AB|BC|MB|NB|NL|NT|NS|NU|ON|PE|QC|SK|YT"
      Return st.Length = 2 AndAlso allProv.IndexOf("|" & st) <> -1
    End Function

    Public Function IsUSAState(ByRef st As String) As Boolean
      Dim allStates = "|AL|AK|AZ|AR|CA|CO|CT|DE|FL|GA|HI|ID|IL|IN|IA|" & _
        "KS|KY|LA|ME|MD|MA|MI|MN|MS|MO|MT|NE|NV|NH|NJ|NM|NY|NC|ND|OH|" & _
        "OK|OR|PA|RI|SC|SD|TN|TX|UT|VT|VA|WA|WV|WI|WY"
      Return st.Length = 2 AndAlso allStates.IndexOf("|" & st) <> -1
    End Function
```

```
    Public Function IsUSAPostalCode(ByVal zip As String) As Boolean
      If String.IsNullOrEmpty(zip) Then Return False

      Return Regex.IsMatch(zip, "^\d{5}(-\d{4})?$")
    End Function

    Public Function IsCanadianPostalCode(ByVal zip As String) As Boolean
      If String.IsNullOrEmpty(zip) Then Return False

      Return Regex.IsMatch(zip, "^[A-Z]\d[A-Z] \d[A-Z]\d$")
    End Function

    Public Function IsNorthAmericanPhone(ByRef phone As String) As Boolean
      If String.IsNullOrEmpty(phone) Then Return False

      Dim expr As String = "^(?:\([2-9]\d{2}\)\ ?|[2-9]\d{2}" & _
                    "(?:\-?|\ ?))[2-9]\d{2}[- ]?\d{4}$"

      Return Regex.IsMatch(phone, expr)
    End Function

    Public Function IsEmail(ByRef email As String) As Boolean
      If String.IsNullOrEmpty(email) Then Return False

      Dim localPartCharset = "[0-9a-zA-Z!#$%*/?|\^{}`~&'+\-=_]"
      Dim domainPartCharset = "[0-9a-zA-Z\-]"
      Dim expr = String.Format("^{0}+(\.{0}+)*@{1}+(\.{1}+)*", _
        localPartCharset, domainPartCharset)

      Return Regex.IsMatch(email, expr)
    End Function
  End Module
End Namespace
```

عليك أن تكون حذرا عندما تقوم بإنشاء قالب My Extension يمكن أن يستخدم في تعريف أي مشروع فيجب عليك الانتباه بشكل خاص إلى القيم المحتملة لـ Option Explicit و Option Strict و Option Compare و Option Infer عندها يمكنك كتابة كودك ليعمل ضمن التعريف الأقل مرونة أو تحديد كل تعريف بشكل خاص في بداية كل ملف كود في توسعتك الذي يعتبر الأمر الأفضل.

كما يجب أن تأخذ إمكانية الاستيرادات المختلفة في المشاريع لتجنب التضارب في الرموز المعرفة على مستوى المشروع وهنا عليك تعديل أي رموز غير معرفة في قالبك بالكلمة المحجوزة Global أو My فمثلا بدلا من استخدام Text.Encoding استخدم Global.Text.Encoding فإن استخدمت Text.Encoding فقط فلن يعمل قالبك في تطبيقات Windows Forms التي ستستورد عندها مجالي أسماء مسميان Text فإن حذفت Global واستخدمت فقط System.Text.Encoding فربما لن يعمل قالبك الذي يحدد مجال أسماء يدعى System. ويعتبر اختبار كودك جيدا قبل توزيعه على الآخرين من الأفكار الجيدة ولا تشذ My Extensions عن هذه القاعدة.

دعنا نستخدم الـ Module التي تم إنشاؤها آليا باسم Module1 لكتابة تجربة لـ My Extensions ويجدر الانتباه إلى انه سيكون من الصعب إعادة استيراد التوسعات عندما تجد شائبة bug لذا عليك محاولة جعل التوسعة تعمل بشكل كامل قدر الإمكان والتأكد من أنها مختبرة جيدا قبل أن تقوم بتصديرها للمرة الأولى. قم بفتح Module1.vb وذلك بالنقر المزدوج عليها في Solution Explorer واستبدل الكود الموجود بالكود التالي ثم اضغط F5 لتشغيل المشروع

```
Module Module1
    Private testNumber As Integer
    Private testGroup As String

    Sub StartTestGroup(ByVal name As String)
        testGroup = name
        testNumber = 0
    End Sub

    Sub Test(ByVal result As Boolean, ByVal expected As Boolean)
        testNumber += 1

        If result <> expected Then
            Console.WriteLine("Test #{0} of group {1} FAILED.", _
              testNumber, testGroup)
        End If
    End Sub

    Sub Main()

        StartTestGroup("IsAlphaNumeric")
        Test(My.Validation.IsAlphaNumeric("123HH2", True, True), True)


        StartTestGroup("IsCanadianPostalCode")
        Test(My.Validation.IsCanadianPostalCode("H0H 0H0"), True)


        StartTestGroup("IsCanadianProvince")
        Test(My.Validation.IsCanadianProvince("ON"), True)


        StartTestGroup("IsEmail")
        Test(My.Validation.IsEmail("a@b.com"), True)

        Console.WriteLine _
          ("Tests are finished. No FAILED messages indicates success.")
        Console.WriteLine("Press any key to continue.")

        Console.ReadKey(True)
    End Sub
End Module
```

توضيب التوسعات كقوالب

بعد أن تمت كتابة التوسعة واختبرت بشكل كامل حان الوقت لتوضيبها كقالب My Extension. ولعمل ذلك اترك المشروع مفتوحا واختر الأمر Export Template من القائمة File فيظهر لك معالج Export Template حيث سنختار item Template من نافذة المعالج كما يتوفر لك خيار في الأسفل لاختيار اسم المشروع الذي سنصدر القالب منه ثم اضغط next ومن قائمة Item to export اختر MyValidation.vb ثم اضغط next ثم عليك اختيارا جميع المراجع الضرورية لعمل قالبك ومن أجل هذا المشروع اختر

Automatically import the وقم بإزالة الإشارة عن الخيار Template Description ثم املأ next ثم اضغط System.Core template into Visual Studio الذي يجعل بيئة التطوير لا تقوم باستيراد القالب مباشرة بعد تصديره ثم اضغط Finish

My Extensions as Templates

في قالب My Extension كما هو الحال في جميع قوالب Visual Studio تكون جميع محددات القالب متوفرة وهي عبارة عن سلاسل نصية خاصة يمكن إدراجها آليا ضمن كود My Extension عندما يتم إضافتها للمشروع فـ $clrversion$ يتم استبدالها برقم النسخة الحالية لـ clr وفي حالة القالب Mu Extension سيتم استبدال $safeitemname$ بقيمة العنصر <DefaultName> في الملف الذي يحمل اللاحقة vstemplate والذي يساوي اسم الملف الذي يحوي كود القالب بدون لاحقته.

وأي شئ آخر يمكن عمله في قالب نظامي يمكنك عمله في القالب My Extension فضبط القيمة <Hidden> في الملف vstemplate إلى False يمكنك من إضافة القالب إلى صندوق الحوار Add New Item عندما تضيف العناصر للمشروع كما يمكنك إجبار إضافة مجمعات للمشروع عند إضافة القالب وذلك ضمن القسم <References> ويمكن إضافة عائلة من ملفات الكود عند انتهائها كالتي يتم إنشاؤها من قبل محرر نماذج ويندوز ذا بالإضافة إلى كود المستخدم.

كما توجد بعض الاختلافات الأساسية بين القوالب الأساسية وقوالب My Extensibility فلكي تمكن قالب العنصر Item Template كي يتم إدارتها من قبل صفحة My Extensions في مصمم مشاريع Visual Basic يجب عليك القيام بعدة خطوات لتعريفها بشكل ملائم.

في البداية يجب عليك إضافة العنصر <CustomDataSignature> للملف MyTemplate.vstemplate المتواجد في ملف zip الذي تم تصديره الذي يشير إلى أن قالب العنصر يجب معاملته كـ My Extension كما يجب عليك منع القالب من الفتح بشكل آلي عندما يتم إضافته للمشروع وذلك بإضافة بعض الصفات للعنصر <ProjectItem> بسبب أن كود MyExtension يجب أن يكون غير مرئي للمستخدم كما يجب عليك إضافة العنصر <Hidden> للملف MyTemplate.vstemplate كي لا يظهر القالب ضمن قائمة القوالب المثبتة عندما يختار المستخدم إضافة عناصر لمشروعه من خلال صندوق الحوار Add New Item وأخيرا عليك إضافة ملف باللاحقة customdata للرزمة وذلك لتوفير معلومات لا تنطبق على القوالب القياسية مثل unique Id و Version number و trigger assembly. ولعمل هذا عليك فك الملف ذو اللاحقة zip والقيام بالتعديلات المناسبة ثم إعادة ضغط الملفات الضرورية ثانية.

في البداية قم بفك محتويات القالب الذي قمنا بتصديره سابقا إلى مجلد فارغ ثم افتح الملف myTemplate.vstemplate وقم بالتعديلات كما في الكود

```
<VSTemplate Version="2.0.0"
  xmlns=http://schemas.microsoft.com/developer/vstemplate/2005
  Type="Item">
 <TemplateData>
   <DefaultName>MyValidation.vb</DefaultName>
   <Name>MyValidation</Name>
   <Description>Extends the My namespace with a set of string
     validation methods.</Description>
   <ProjectType>VisualBasic</ProjectType>
   <SortOrder>10</SortOrder>
   <Icon>TemplateIcon.ico</Icon>

   <!-- Configures the template to not be shown as an option
     when the user selects to add an item template to his project -->
   <Hidden>true</Hidden>

   <!-- Indicates it is to be treated as a My Extension. -->
   <CustomDataSignature>Microsoft.VisualBasic.MyExtension
   </CustomDataSignature>
```

```
  </TemplateData>
  <TemplateContent>
    <References>
      <Reference> <Assembly>System.Core</Assembly> </Reference>
    </References>
    <ProjectItem SubType="Code" TargetFileName="$fileinputname$.vb"
      ReplaceParameters="true" OpenInEditor="false"
      OpenInWebBrowser="false" OpenInHelpBrowser="false">MyValidation.vb
    </ProjectItem>
  </TemplateContent>
</VSTemplate>
```

أنشئ ملفا جديدا - سوف نضيفه للرزمة لاحقا - وسمه MyValidation.customdata وافتحه بواسطة الـ Notepad وأضف إليه الكود التالي

```
<VBMyExtensionTemplate
    ID="MyValidationMyNamespaceExtension"
    Version="1.0.0.0" />
```

والواصفة AssemblyFullName التي تم حذفها في هذا المثال تشير إلى أنه عندما يتم إضافة هذا المجمع للمشروع فسوف يتم دعوة المستخدم لإضافة My Extension وعندما يزال المجمع من المشروع فسوف يتم دعوة المستخدم لإزالة My Extension ولإضافة مجمع قدح Trigger Assembly يمكن إضافة قسم كالتالي للملف customdata

```
<VBMyExtensionTemplate
    ID="MyValidationMyNamespaceExtension"
    Version="1.0.0.0"
    AssemblyFullName="System.Windows.Forms" />
```

قم الآن بإعادة نسخ جميع الملفات لداخل الملف zip باستخدام Windows Explorer

الملف vbproj

التعامل مع My Extension هو عملية مخفية آلية ولكي يقوم Visual Studio بمعالجتها بسلاسة باستخدام دخل قليل من المستخدم أصبح لملف المشروع الذي يحتوي على التوسعة بعض الصفات الجديدة التي يمكنك رؤيتها إذا قمت بفتح الملف ذو اللاحقة vbproj الخاص بالمشروع الذي قمت بإضافة التوسعة MyValidation إليه بأي محرر نصي فسوف ترى شيئا شبيها بالتالي

```
<ItemGroup>
  ...
  <Compile Include="My Project\MyExtensions\MyValidation.vb">
    <VBMyExtensionTemplateID>
      MyValidationMyNamespaceExtension
    </VBMyExtensionTemplateID>
    <VBMyExtensionTemplateVersion>
      1.0.0.0
    </VBMyExtensionTemplateVersion>
  </Compile>
  ...
</ItemGroup>
```

ولكن إذا ألقيت نظرة على القسم الخاص بملف كود أساسي مثل الملف Module1.vb فسوف ترى

```
<ItemGroup>
  <Compile Include="Module1.vb" />
</ItemGroup>
```

وكما ترى هناك بعض الاختلافات فالملف MyValidation.vb سوف يتم تخزينه في مجلد المشروع وهذا يعني أنه سوف يكون غير مرئي إلا إذا فعلت خيار Show All items في Solution Explorer وستجد أن له قطعتان إضافيتان من البيانات توافق القيم المضافة في الملف MyValidation.customdata. وبتخزين هذه المعلومات يمكن لـ Visual studio أن يسمح للمستخدم بإضافة وإزالة التوسعات من صفحة My Extensions في خصائص المشروع بدلا عن إضافة الملف بشكل يدوي وإن كان هناك مؤشر لمجمع قادح وتمت إزالته فسيقوم لـ Visual Studio بتحذير المستخدم من أن My Extension لن يستطيع العمل ويعطيه خيارا لإزالته كما تمنع من تواجد أكثر من نسخة من التوسعة في نفس المشروع.

جعل صندوق النصوص يقبل العمليات الحسابية بدون استخدام الخاصية Text ودوال تحويل الأنواع

يعتبر هذا الموضوع تطبيقا على مفهوم بعنوان التحميل الزائد للمعاملات Operators Overloading فيرجى مراجعة الموضوع المذكور إن لك يكن لديك إطلاع عليه

فإن كنت تريد القيام بعملية ضرب قيمة ما يوجد داخل صندوق النصوص بـ10 مثلا فللجميع سيقول أن بيئة التطوير لن تقبل الكود التالي

```
a = TextBox1 * 10
```

وفي مقالي هذا سأقوم بإقناع بيئة التطوير بقبوله تطبيقا على مفهوم التحميل الزائد للمعاملات
أنشئ مشروعا جديدا وضع فيه صندوق نصوص و زر أوامر فقط ثم من قائمة Project اختر الأمر Add Class و سم الفئة الجديدة MyTextBox حيث سنقوم بوراثتها من صندوق النصوص العادي المستخدم ثم أدخل الكود التالي في الفئة الجديدة الذي سيكون الكود الكامل للفئة

```
Public Class MyTextBox
    Inherits TextBox

    Public Shared Operator *(ByVal Op1 As MyTextBox, ByVal op2 As Integer) As
Integer
        Return (CInt(Op1.Text) * op2)
    End Operator

    Public Shared Operator *(ByVal op2 As Integer, ByVal Op1 As MyTextBox) As
Integer
        Return (CInt(Op1.Text) * op2)
    End Operator

End Class
```

اعمل build للمشروع

في Solution Explorer اضغط الزر Show All Files ثم وسع العقدة Form1.vb وافتح محرر الكود الخاص بالملف Form1.Designer.vb واستبدل النص System.Windows.Forms.TextBox أينما ورد في الملف المذكور باسم فئتنا الجديدة MyTextBox ثم قم بالحفظ وأغلق كافة الملفات المفتوحة ثم قم بفتح محرر النماذج الخاص بـ Form1 وانقر نقرا مزدوجا على الزر ليتم إضافة معالج حدث للنقر على زر الأوامر وضع فيه الكود التالي

```
Dim a As Integer = TextBox1 * 34
Dim b As Integer = 65 * TextBox1

MsgBox(a.ToString & vbCrLf & b.ToString)
```

سترى نتيجة الكود الذي وضعناه في فئتنا الجديدة أن صندوق النصوص أصبح يقبل عملية الضرب مباشرة بدون استخدام الخاصية Text و دوال التحويل بين الأنواع

وهذه فئة موسعة عن الفئة المذكورة سابقا تتضمن القيام بالعمليات الحسابية الأربع ولنوع البيانات integer حيث يمكنك عمل إجرائيات كي تنفذ العمليات الحسابية على أي نوع تريده

```
Public Class MyTextBox
    Inherits TextBox

    Public Shared Operator *(ByVal Op1 As MyTextBox, ByVal op2 As Integer) _
                             As Integer

        Return (CInt(Op1.Text) * op2)
    End Operator

    Public Shared Operator *(ByVal op2 As Integer, ByVal Op1 As MyTextBox) _
                             As Integer

        Return (CInt(Op1.Text) * op2)
    End Operator

    Public Shared Operator +(ByVal a As MyTextBox, ByVal b As Integer) _
                             As Integer

        Return (CInt(a.Text) + b)
    End Operator

    Public Shared Operator +(ByVal b As Integer, ByVal a As MyTextBox) _
                             As Integer

        Return (CInt(a.Text) + b)
    End Operator

    Public Shared Operator -(ByVal a As MyTextBox, ByVal b As Integer) _
                             As Integer

        Return (CInt(a.Text) - b)
    End Operator

    Public Shared Operator -(ByVal b As Integer, ByVal a As MyTextBox) _
                             As Integer

        Return (b - CInt(a.Text))
    End Operator

    Public Shared Operator /(ByVal a As MyTextBox, ByVal b As Integer) As Double
        Return (CInt(a.Text) / b)
    End Operator

    Public Shared Operator /(ByVal b As Integer, ByVal a As MyTextBox) As Double
        Return (b / CInt(a.Text))
    End Operator

End Class
```

القسم التاسع - تعدد المسارات

ويحتوي على الموضوعات التالية:

- Threading in Windows Forms Applications
- استخدام بحيرة المسارات Using the Thread pool
- تزامن المسارات Thread Synchronization
- كيفية تنفيذ عملية في مسار آخر وإظهار النتيجة في التحكمات على النموذج

Threading in Windows Forms Applications

تكمن المشكلة في أغراض Windows Forms هو أن التحكمات والنموذج ذات نفسه هو أنه يجب الوصول إليهم حصريا من خلال المسار الذي قام بإنشائهم وفي الحقيقة كل أغراض Windows Forms تعتمد على STA Model وذلك بسبب أنها جميعا معتمدة على هيكلية رسائل Win32 والتي ترث مسارات الغرفة Apartment-Threaded مما يعني أنه يمكنك إنشاء النموذج أو التحكم على أي مسار تريده ولكن جميع الطرق المرتبطة به يجب استدعاؤها من نفس المسار. مما يؤدي إلى ظهور العديد من المشاكل بسبب أن أقسام الدوت نيت الأخرى تستخدم Free-Threading model ومزج كلا النوعين بدون حكمة تعتبر فكرة سيئة وحتى لو لم تقم بإنشاء مسار بشكل واضح في كودك ربما ستظهر لك بعض المشاكل في جميع الأحوال فمثلا عندما تحاول الوصول إلى عنصر واجهة مستخدم UI Element من خلال الطريقة Finalize لنوع ما ونحن نعلم أن الطريقة Finalize يتم تنفيذها على مسار مختلف عن المسار الرئيسي

The ISynchronizeInvoke Interface

عناصر التحكم الوحيدة التي يمكنك استدعاؤهم من مسار آخر هم الذين يتم عرضهم من خلال الواجهة ISynchronizeInvoke التي تمتلك الطرائق BeginInvoke و EndInvoke و الخاصية InvokeRequired القابلة للقراءة فقط. حيث تعيد الخاصية InvokeRequired القيمة True إذا كان المستدعي لا يستطيع الوصول إلى التحكم مباشرة وذلك عندما يعمل المستدعي على مسار مختلف عن المسار الذي تم إنشاء التحكم فيه ففي هذه الحالة يحب على المستدعي استدعاء الطريقة Invoke للوصول إلى أي عنصر خاص بالتحكم وهذه الطريقة متزامنة لهذا يتم إيقاف المسار المستدعي حتى يكمل مسار UI تنفيذ الطريقة. أو يمكن للمسار المستدعي استخدام الطرائق BeginInvoke و EndInvoke لتنفيذ العملية بشكل لا متزامن.
تأخذ الطريقة Invoke إجراء مفوض يشير إلى طريقة (Sub أو Function) ويمكنه أخذ مصفوفة من النوع Object كمحدد ثاني إذا كانت الطريقة تتوقع واحد أو أكثر من المحددات وتضمن هيكلية نماذج ويندوز أن الإجراء الذي يشير إليه المفوض يتم تنفيذه في المسار UI لهذا يمكنه بأمان الوصول إلى أي تحكم على النموذج.
سنرى كيف يمكننا استخدام الطريقة Invoke للوصول إلى تحكم من مسار غير المسار UI حيث يظهر لنا المثال التالي كيف يمكننا زيارة جميع المجلدات ضمن شجرة مجلد من مسار ثانوي بينما يتم إظهار السم المجلد في تحكم Label وأول شئ سنقوم بعمله هو تحديد طريقة تقوم بعمل الإظهار المطلوب التي يمكنها أن تكون مجرد إجراء بسيط

```
' This method must run in the main UI thread.
Sub ShowMessage(ByVal msg As String)
   Me.lblMessage.Text = msg
   Me.Refresh()
End Sub
```

ثم نقوم بتحديد إجراء مفوض يشير لتلك الطريقة ومتغير يحمل كائن لذلك المفوض يكون معرفا على مستوى النموذج كي تتم مشاركته بين جميع الطرائق ضمن النموذج

```
' A delegate that can point to the ShowMessage procedure
Delegate Sub ShowMessageDelegate(ByVal msg As String)
' An instance of the delegate
Dim threadSafeDelegate As ShowMessageDelegate
```

وستحتاج لطريقة تبدأ المسار الثانوي مثلا إجراء معالجة الحدث Click لزر أوامر Button

```
' Parse the c:\Windows directory when the user clicks this button.
Private Sub btnSearch_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnSearch.Click
   Dim t As New Thread(AddressOf SearchFiles)
   t.Start("c:\windows")
End Sub
```

وأخيرا تقوم بكتابة الكود الذي سيعمل على المسار الثانوي حيث أنه من الضروري لذلك الكود أن يستطيع الوصول للتحكم lblMessage باستدعاء الطريقة ShowMessage وهذا يتم من خلال الطريقة Invoke في فئة النموذج Form Class أو الطريقة Invoke لأي تحكم موجود على النموذج والتي تكون مكافئة لها تماما

```
' (This method runs in a non-UI thread.)
```

```
Sub SearchFiles(ByVal arg As Object)
   ' Retrieve the argument.
   Dim path As String = arg.ToString()
   ' Prepare the delegate
   threadSafeDelegate = New ShowMessageDelegate(AddressOf ShowMessage)
   ' Invoke the worker procedure. (The result isn't used in this demo.)
   Dim files As List(Of String) = GetFiles(path)
   ' Show that execution has terminated.
   Dim msg As String = String.Format("Found {0} files", files.Count)
   Me.Invoke(threadSafeDelegate, msg)
End Sub

' A recursive function that retrieves all the files in a directory tree
' (This method runs in a non-UI thread.)
Function GetFiles(ByVal path As String) As List(Of String)
   ' Display a message.
   Dim msg As String = String.Format("Parsing directory {0}", path)
   Me.Invoke(threadSafeDelegate, msg)
   ' Read the files in this folder and all subfolders.
   Dim files As New List(Of String)
   For Each fi As String In Directory.GetFiles(path)
      files.Add(fi)
   Next
   For Each di As String In Directory.GetDirectories(path)
      files.AddRange(GetFiles(di))
   Next
   Return files
End Function
```

وستتعقد العملية أكثر إن احتجنا لاستخدام الطريقة ShowMessage على جميع المسارات فالطريقة GetFiles مثلا يمكن استدعاؤها من المسار UI وفي هذه الحالة عمل الاستدعاء باستخدام الطريقة Invoke يضيف استباقا للأمور يجب تجنبه لذلك يجب علينا فحص قيمة الخاصية InvokeRequired واستخدام الطريقة العادية إن كانت تعيد القيمة False

```
' (Inside the SearchFiles and GetFiles methods)
If Me.InvokeRequired Then
   Me.Invoke(threadSafeDelegate, msg)
Else
   ShowMessage(msg)
End If
```

والطريقة الأفضل من ذلك بدلا من فحص الخاصية InvokeRequired من أجل كل مستدعي سنقوم بفحصها من داخل الطريقة ShowMessage

```
' This method can run in the UI thread or in a non-UI thread.
Sub ShowMessage(ByVal msg As String)
   ' Use the Invoke method only if necessary.

   If Me.InvokeRequired Then
      Me.Invoke(threadSafeDelegate, msg)
      Return
   End If

   Me.lblMessage.Text = msg
   Me.Refresh()
End Sub
```

فبعد هذا التغيير أي قطعة من الكود ستحتاج لإظهار رسالة على التحكم lblMessage ستحتاج فقط لاستدعاء ShowMessage بدون القلق حول أي مسار يتم تنفيذ الكود عليه
وفي بعض الظروف في فيجول بايزيك 2005 أو الفريمورك رقم 2 يقوم التطبيق بالوصول للتحكم عن طريق مسار غير مسار الإظهار non-UI thread بدون التسبب بأية مشاكل فيمكن حدوث ذلك مثلا عندما تحاول الوصول إلى تحكمات بسيطة مثل Label أو عندما تقوم

بعمليات لا تسبب إرسال رسائل Win32 في الخلفية كما أن العديد من الخصائص يمكن قراءتها وليس تعديلها بدون التسبب بمشاكل وذلك لأن قيمة تلك الخصائص مخزنة في عنصر ضمن تحكم الدوت نيت

The BackgroundWorker Component

على الرغم من أن الواجهة ISynchronizeInvoke تجنبك من الوقوع في المشاكل المتعلقة بالمسارات في تطبيقات نماذج ويندوز يحتاج معظم مطوري فيجول بايزيك لطريقة أفضل وأقل أخطاء فأنت تحتاج مثلا لطريقة بسيطة لإلغاء طريقة غير متزامنة بأسلوب آمن الشئ الذي لا توفره الواجهة المذكورة بشكل تلقائي. ومن أجل هذا السبب قامت مايكروسوفت بإضافة المكون BackgroundWorker إلى صندوق الأدوات واستخدامه سهل جدا مما يسهل عملية إنشاء تطبيقات ويندوز متعددة المسارات.

يمتلك المكون BackgroundWorker خاصيتان مثيرتان للاهتمام فالخاصية WorkerReportsProgress تكون قيمتها True إذا أطلق المكون الحدث ProgressChanged والخاصية WorkerSupportsCancellation تكون قيمتها True إذا كان المكون يدعم الطريقة CancelAsync وتكون القيمة الافتراضية لكلا الطريقتين False لذا يجب عليك ضبط قيمتهم إلى True إذا أردت الاستفادة من جميع مزايا هذا التحكم والمثال الذي سيطرح هنا يفترض أنه قد تم ضبط كلتا القيمتين إلى True ويتطلب استخدام المكون BackgroundWorker بشكل عام العمليات التالية:

١. إنشاء إجراء معالجة للحدث DoWork وملؤها بالكود الذي تريد أن يتم تنفيذه على المسار الثانوي ويتم تشغيل هذا الكود عندما يتم استدعاء الطريقة RunWorkerAsync وهي تقبل محددا يتم تمريره لإجراء معالجة الحدث DoWork حيث لا يمكن للكود الموجود هناك الوصول مباشرة للتحكمات على النموذج لأنه يعمل في مسار آخر

٢. استخدم الطريقة ReportProgress من داخل الحدث DoWork عندما تريد الوصول إلى عنصر على النموذج وهذه الطريقة تطلق الحدث ProgressChanged إذا كانت قيمة الخاصية Worker-ReportsProgress هي True وإلا سيتم إطلاق استثناء Worker-ReportsProgress في حالة كون قيمتها False والكود في إجراء معالجة الحدث ProgressChanged يعمل في نفس المسار UI ولهذا يمكنه الوصول بأمان لأي من تحكمات النموذج

٣. استخدم الطريقة CancelAsync للتحكم BackgroundWorker لإيقاف المسار الثانوي مباشرة وهذه الطريقة تستدعي ضبط الخاصية WorkerSupportsCancellation إلى True وإلا سيتم إطلاق استثناء InvalidOperationException في حالة كون قيمتها False ويجب على الكود في DoWork التحقق دوريا من الخاصية CancellationPending والخروج بأمان عندما تصبح قيمتها True

٤. كتابة إجراء معالجة للحدث RunWorkerCompleted إن كنت تريد القيام بأية أعمال عندما ينتهي عمل المسار الثانوي إما بشكل طبيعي أو بواسطة الإلغاء والكود في إجراء معالجة هذا الحدث يعمل في المسار UI لذا يستطيع الوصول لجميع عناصر النموذج

وبشكل عام فالكود في معالج الحدث DoWork يجب أن يعيد قيمة للمسار الأساسي بدلا من تعيين هذه القيمة في حقل على مستوى الفئة فعلى الكود تعيين هذه القيمة للخاصية Result للغرض DoWorkEventArgs فتكون هذه القيمة متوفرة للمسار الأساسي بواسطة الخاصية Result للغرض RunWorkerCompletedEventArgs الممرر للحدث RunWorkerCompleted وهذا كود نموذجي يستخدم العنصر BackgroundWorker

```
' The button that starts the asynchronous operation
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStart.Click
   Dim argument As Object = "abcde"            ' The argument

   BackgroundWorker1.RunWorkerAsync(argument)
   ' Disable this button, and enable the "Stop" button.
   btnStart.Enabled = False
   btnStop.Enabled = True
End Sub

' The button that cancels the asynchronous operation
Private Sub btnStop_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStop.Click
   BackgroundWorker1.RunWorkerAsync(argument)
End Sub
```

```
' The code that performs the asynchronous operation
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
      ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork
   ' Retrieve the argument.
   Dim argument As Object = e.Argument
   Dim percentage As Integer = 0
   …
   ' The core of the asynchronous task
   Do Until BackgroundWorker1.CancellationPending
      …
      ' Report progress when it makes sense to do so.
      BackgroundWorker1.ReportProgress(percentage)
   Loop
   ' Return the result to the caller.
   e.Result = primes
End Sub

' This method runs when the ReportProgress method is invoked.
Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
             ByVal e As ProgressChangedEventArgs) Handles _
             BackgroundWorker1.ProgressChanged

   ' It is safe to access the user interface from here.
   ' For example, show the progress on a progress bar or another control.
   ToolStripProgressBar1.Value = e.ProgressPercentage
End Sub

' This method runs when the asynchronous task is completed (or canceled).
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
                ByVal e As RunWorkerCompletedEventArgs) Handles _
                BackgroundWorker1.RunWorkerCompleted
   ' It is safe to access the user interface from here.
   …
   ' Reset the Enabled state of the Start and Stop buttons.
   btnStart.Enabled = True
   btnStop.Enabled = False
End Sub
```

يظهر لك المثال التالي كيف يمكن استخدام العنصر BackgroundWorker للبحث عن الملفات في مسار غير متزامن وهي نفس المشكلة التي طرحت عند الحديث عن The ISynchronizeInvoke Interface في هذا الموضوع سابقا وبهذا يمكنك مقارنة الطريقتين بسهولة. وستكون النسخة الجديدة المعتمدة على BackgroundWorker أكثر تعقيدا بقليل بسبب أنها تدعم الإلغاء لعمل غير متزامن

```
' The result from the SearchFiles procedure
Dim files As List(Of String)
' We need this variable to avoid nested calls to ProgressChanged.
Dim callInProgress As Boolean

' The same button works as a Start and a Stop button.
Private Sub btnStart_Click(ByVal sender As Object, ByVal e As EventArgs) _
      Handles btnStart.Click
   If btnStart.Text = "Start" Then
      lstFiles.Items.Clear()
      Me.BackgroundWorker1.RunWorkerAsync("c:\windows")
      Me.btnStart.Text = "Stop"
   Else
      Me.BackgroundWorker1.CancelAsync()
   End If
End Sub

' The code that starts the asynchronous file search
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
```

```
        ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork

   ' Retrieve the argument.
   Dim path As String = e.Argument.ToString()
   ' Invoke the worker procedure.
   files = New List(Of String)
   SearchFiles(path)
   ' Return a result to the RunWorkerCompleted event.
   Dim msg As String = String.Format("Found {0} files", files.Count)
   e.Result = msg
End Sub

' A recursive function that retrieves all the files in a directory tree.
Sub SearchFiles(ByVal path As String)
   ' Display a message.

   Dim msg As String = String.Format("Parsing directory {0}", path)
   ' Notice that we don't really use the percentage;
   ' instead, we pass the message in the UserState property.
   Me.BackgroundWorker1.ReportProgress(0, msg)

   ' Read the files in this folder and all subfolders.
   ' Exit immediately if the task has been canceled.
   For Each fi As String In Directory.GetFiles(path)
      If Me.BackgroundWorker1.CancellationPending Then Return
      files.Add(fi)
   Next
   For Each di As String In Directory.GetDirectories(path)
       If Me.BackgroundWorker1.CancellationPending Then Return
       SearchFiles(di)
   Next
End Sub


Private Sub BackgroundWorker1_ProgressChanged(ByVal sender As Object, _
      ByVal e As ProgressChangedEventArgs) _
      Handles BackgroundWorker1.ProgressChanged
   ' Reject nested calls.
   If callInProgress Then Return
   callInProgress = True
   ' Display the message, received in the UserState property.
   Me.lblMessage.Text = e.UserState.ToString()
   ' Display all files added since last call.
   For i As Integer = lstFiles.Items.Count To files.Count - 1
      lstFiles.Items.Add(files(i))
   Next
   Me.Refresh()
   ' Let the Windows operating system process message in the queue.
   ' If you omit this call, clicks on buttons are ignored.
   Application.DoEvents()
   callInProgress = False
End Sub

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
      ByVal e As RunWorkerCompletedEventArgs) _
      Handles BackgroundWorker1.RunWorkerCompleted
   ' Display the last message and reset button's caption.
   Me.lblMessage.Text = e.Result.ToString()
   btnStart.Text = "Start"
End Sub
```

والكود هنا يشرح نفسه ماعدا إجراء الحدث ProgressChanged حيث يجب أن يتضمن الكود استدعاء للطريقة Application.DoEvents وإلا لن يتمكن التطبيق من معالجة الأحداث المنطلقة مثل حدث النقر على الزر Stop أو أي عمل آخر ممكن

إضافته للواجهة ومع ذلك فاستدعاء هذه الطريقة سيسبب استدعاءات معششة للإجراء ProgressChanged مما قد يسبب إطلاق استثناء StackOverflowException ومن أجل عدم حدوث هذا يتم استخدام حقل منطقي مساعد callInProgress لتجنب حدوث مثل هذه الاستدعاءات المعششة
لاحظ أيضا أن هذا التطبيق لا يحتاج للإعلام عن نسبة التقدم للمسار الرئيسي ويستخدم الطريقة ReportProgress فقط لتنفيذ جزء من الكود في المسار الرئيسي للبرنامج والرسالة الفعلية للإظهار يتم تمريرها للخاصية UserState وإن كان تطبيقك يستخدم progress bar أو أي مؤشر آخر للتقدم يجب عليك تجنب استدعاء الطريقة ReportProgress بدون داعي لأنها تتسبب بتبديل المسارات وتكون مكلفة كثيرا عندما يتعلق الأمر بوقت المعالجة وفي هذه الحالة يجب عليك تخزين مؤشر التقدم في حقل في الفئة واستدعاء الطريقة فقط في حالة حدوث تقدم فعلي

```
Dim currentPercentage As Integer

Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
             ByVal e As DoWorkEventArgs) Handles BackgroundWorker1.DoWork

   Const TotalSteps = 5000
   For i As Integer = 1 To TotalSteps
      …
      ' Evaluate progress percentage.
      Dim percentage As Integer = (i * 100) \ TotalSteps
      ' Report to UI thread only if percentage has changed.
      If percentage <> currentPercentage Then
         BackgroundWorker1.ReportProgress(percentage)
         currentPercentage = percentage
      End If
   Next
End Sub
```

استخدام بحيرة المسارات Using the Thread pool

إنشاء العديد من المسارات قد يسبب انخفاض أداء النظام بسرعة وخاصة عندما تصرف هذه المسارات معظم وقتها في حالة سبات أويعاد تشغيلها بصورة دورية بغرض قراءة مصدر ما أو تحديث الإظهار. ولتحسين أداء كودك يمكنك إعادة ترتيب بحيرة المسارات بشكل يضمن أكفأ استخدام للموارد باستخدام بعض الأغراض Objects الموجودة في مجال الأسماء System.Threading بحيرة المسارات

The ThreadPool Type

يتم إنشاء بحيرة المسارات عندما تقوم باستدعاء الدالة ThreadPool.QueueUserWorkItem والتي تحتاج لإجراء مفوض WaitCallback delegate وغرض Object اختياري يستخدم لتمرير البيانات للمسار والإجراء المفوض يجب أن يشير إلى Sub يمرر له محدد وحيد من النوع Object بحيث تكون قيمته محتوية على البيانات التي نريد تمريرها للمسار أو Nothing عندما لا توجد بيانات نريد تمريرها وقطعة الكود التالية تبين لك كيف يمكنك استخدام عدد كبير من المسارات لاستدعاء إجراء في فئة Class

```
For i As Integer = 1 To 20
     ' Create a new object for the next lightweight task.
      Dim task As New LightweightTask()
      ' Pass additional information to it. (Not used in this demo.)
       task.SomeData = "other data"
       ' Run the task with a thread from the pool.
       ' (Pass the counter as an argument.)
       ThreadPool.QueueUserWorkItem(AddressOf task.Execute, i)
 Next
```

وقطعة الكود التالية تحتوي على الكود الذي يتم تنفيذه فعلا عندما يتم سحب المسار من البركة

```
Class LightweightTask
    Public SomeData As String

    ' The method that contains the interesting code
    ' (Not really interesting in this example)
    Sub Execute(ByVal state As Object)
        Console.WriteLine("Message from thread #{0}", state)
    End Sub
End Class
```

والمسار العامل يمكنه تحديد فيما إذا كان قد أخذ من بحيرة المسارات أم لا بتحري قيمة الخاصية Thread.CurrentThread.IsThreadPoolThread ويمكنك معرفة العدد الأقصى للمسارات في البركة باستدعاء الطريقة الساكنة ThreadPool.GetMaxThreads وعدد المسارات المتاحة حاليا باستدعاء الطريقة الساكنة ThreadPool.GetAvailableThreads . كما تم إضافة طريقة جديدة SetMaxThreads في الفريموورك NET Framework 2.0. 2 تمكنك من تغيير العدد الأقصى للمسارات الموجودة في البركة

```
' Maximum 30 worker threads and maximum 10 asynchronous I/O threads in the pool
ThreadPool.SetMaxThreads(30, 10)
```

في بعض الأحيان قد تحتار في نقطة تساؤل هل أقوم بإنشاء المسار بنفسي أم أستعيره من بحيرة المسارات. وتظهر هنا قاعدة جيدة: استخدم فئة المسارات Thread class عندما تريد تنفيذ عملية تريد تنفيذها بأسرع وقت أو عندما تريد القيام بعملية تستهلك الوقت ولا يتم تنفيذها كثيرا وفي معظم الحالات بشكل عام يجب عليك استخدام بحيرة المسارات.

The Timer Type

تقدم الفريموورك عدة أنواع من المؤقتات كل منها يمتلك نقاط قوته وضعفه. فمثلا يجب عليك استخدام التحكم System.Windows.Forms.Timer عندما تقوم بالعمل على تطبيق من النوع Windows Forms applications وإن لم يكن برنامجك يمتلك واجهة للمستخدم يجب عليك عندها استخدام الفئة System.Threading.Timer أو الفئة System.Timers.Timer وتعتبر هاتان الفئتان متساويتين في العمل تقريبا والشرح التالي على System.Threading.Timer ينطبق أيضا على System.Timers.Timer

الفئة Timer في مجال الأسماء System.Threading يقدم طريقة بسيطة لمؤقت يستدعي إجرائية محددة حيث يمكنك استخدام هذه الفئة لجدولة عمل في وقت معين في المستقبل ويمكن تنفيذه بالتكرار الذي تحتاجه مهما يكن ابتداء من مرة واحدة فما فوق وباني المؤقت يأخذ أربعة محددات:

- إجراء مفوض TimerCallback delegate يشير إلى الإجراء الذي يستدعى عندما ينتهي زمن المؤقت ويجب أن يكون هذا الإجراء من النوع Sub يأخذ محدد واحد من النوع Object
- غرض Object يتم تمريره للإجراء الذي يشير إليه المفوض ويمكن أن يكون من عدة أنواع كسلسلة نصية أو مصفوفة أو مجموعة Collection أو أي نوع بيانات آخر يحتوي على البيانات التي سيتم تمريرها للإجراء وإن لم تكن تحتاج لتمرير قيم استخدم Nothing بكل بساطة
- قيمة من النوع TimeSpan تحدد زمن المؤقت الذي سيتم استدعاء الإجراء بعده كما يمكن تحديدها باستخدام قيمة من النوع Long أو UInteger وفي هذه الحالة يقاس الزمن بالميللي ثانية (1/1000 من الثانية) وعند تمرير Timeout.Infinite كقيمة لا يتم إطلاق المؤقت أبدا أو القيمة 0 صفر لإطلاق المؤقت مباشرة
- قيمة من النوع TimeSpan تحدد زمن المؤقت والتي بدورها تحدد زمن تكرار إطلاق المؤقت بعد المرة الأولى. وهذه أيضا يمكن تحديدها بقيمة من النوع Long أو UInteger وهنا أيضا يصبح الوقت مقاسا بالميللي ثانية ويمكنك تمرير القيمة 1 أو Timeout.Infinite لإطلاق المؤقت مرة واحدة فقط.

وهذه القيم التي تمررها لباني المؤقت غير متوفرة كخصائص. وبعد تشغيل المؤقت يمكنك تغيير هذه القيم فقط باستخدام الطريقة Change method والتي تأخذ محددين يحددان وقت التشغيل وفترة زمن المؤقت ويمتلك Timer object إجراء Stop الذي يقوم بإيقاف المؤقت الذي يتم إيقافه عبر استدعاء الإجراء Dispose وترينا قطعة الكود التالية مثالا عما تحدثنا عنه حول المؤقت

```
Sub TestThreadingTimer()
    ' Get the first callback after one second.
    Dim dueTime As New TimeSpan(0, 0, 1)
    ' Get additional callbacks every half second.
    Dim period As New TimeSpan(0, 0, 0, 0, 500)
    ' Create the timer.
    Using t As New Timer(AddressOf TimerProc, Nothing, dueTime, period)
        ' Wait for five seconds in this demo, and then destroy the timer.
        Thread.Sleep(5000)
    End Using
End Sub

' The callback procedure
Sub TimerProc(ByVal state As Object)
    ' Display current system time in console window.
    Console.WriteLine("Callback proc called at {0}", Date.Now)
End Sub
```

وفي النهاية تجدر ملاحظة أن الإجراء المستدعى يتم تنفيذه على مسار مأخوذ من بركة المسارات لذا يجب عليك التحكم بالمتغيرات والمصادر الأخرى المستخدمة من قبل المسار الرئيسي للبرنامج عبر استخدام ما يدعى بتزامن المسارات

تزامن المسارات Thread Synchronization

The SyncLock Statement

خلال زمن التشغيل لا يوجد شئ يضمن لك أن يسير الكود بشكل نظامي بدون مقاطعات وتكون عملية التشغيل بدون مقاطعات عملية قاسية على نظام التشغيل وخاصة عندما يكون عبارة عن بيئة متعددة المهام وفي معظم الحالات التي ستحتاجها ستكون قانعا بالدقة ضمن البرنامج الواحد وذلك عند معالجة الكود فعلى سبيل المثال يكون كافيا لك ضمان أن مسار تنفيذ واحد ضمن التطبيق الحالي يستطيع تنفيذ قطعة معينة من الكود في وقت محدد ويمكنك تحقيق ذلك بتضمين قطعة الكود تلك ضمن كتلة SyncLock…End SyncLock والذي يحتاج إلى متغير كمحدد له محققا المتطلبات التالية:

- يجب أن يكون مشترك بين جميع المسارات ويكون في العادة متغير على مستوى الفئة وبدون الخاصية ThreadStatic
- يجب أن يكون من نوع مرجعي مثل String أو Object واستخدام أنواع القيمة ينتج عنه خطأ في الترجمة
- يجب أن لا يحتوي على القيمة Nothing وفي حال تمرير القيمة Nothing سيسبب أخطاء في زمن التنفيذ

وفيما يلي مثال عن كتلة SyncLock

```
' The lock object. (Any non-Nothing reference value will do.)
Private consoleLock As New Object()

Sub SynchronizationProblem_Task(ByVal obj As Object)
   Dim number As Integer = CInt(obj)
   ' Print a lot of information to the console window.
   For i As Integer = 1 To 1000
      SyncLock consoleLock
         ' Split the output line in two pieces.
         Console.Write(" ")
         Console.Write(number)
      End SyncLock
   Next
End Sub
```

والكود السابق يستخدم المتغير consoleLock للتحكم بالوصول للغرض Console وهو يشكل المصدر الوحيد المشترك بين جميع المسارات في المثال ولهذا فهو المصدر الذي يجب عليك تحقيق التزامن من أجله والتطبيقات الحقيقية يمكن أن تحوي العديد من كتل SyncLock والتي يمكن أن تستخدم نفس المتغير المحلي أو عدة متغيرات مختلفة من أجل اختلاف البصمة وهنا يجب عليك استخدام متغيرا مميزا من أجل كل نوع من أنواع المصادر المشتركة التي يجب عمل التزامن من اجلها أو من أجل مجموعة التعابير التي يجب تنفيذها ضمن المسار في نفس الوقت.

وعندما تستخدم كتلة SyncLock يتضمن الكود تلقائيا كتلة Try…End Try مخفية من أجل ضمان تحرير القفل بشكل صحيح إذا تم إطلاق استثناء ومن أجل هذا لا يمكنك القفز لعبارة داخل الكتلة SyncLock. وإن كانت الكتلة SyncLock موضوعة داخل إجراء خاص بتواجد Instance لفئة ما وجميع المسارات العاملة ضمن إجراء في ذلك التواجد Instance للفئة يمكنك تمرير Me لعبارة الـ SyncLock وذلك بسبب أن هذا الغرض يحقق كل المتطلبات (يمكن الوصول إليه من جميع المسارات – وهو قيمة مرجعية – وبالتأكيد هو ليس Nothing)

```
Class TestClass
   Sub TheTask()
      SyncLock Me
         ' Only one thread at a time can access this code.
         …
      End SyncLock
   End Sub
End Class
```

ملاحظة: يمكنك استخدام Me بهذه الطريقة فقط إن كنت تريد عمل التزامن على مصدر وحيد كملف محدد مثلا أو نافذة الكونسول Console Window وإن كان لديك عدة كتل تزامن التي تحمي عدة مصادر ستستخدم بشكل تلقائي عدة متغيرات كمحددات لكتلة SyncLock. والشئ

الذي له أهمية أكبر مما ذكر هو أنه يجب عليك استخدام Me كمحدد فقط إذا كانت الفئة غير مرئية خارج المجمع الحالي عدا ذلك يمكن لتطبيق آخر استخدام نفس التواجد Instance للفئة ضمن كتلة SyncLock مختلفة وبهذا فلن يتم تنفيذ عدة كتل من الكود بدون سبب حقيقي محدد وبشكل عام لا يجب عليك استخدام غرض Object عام مرئي من مجمعات أخرى كمحدد لكتلة SyncLock. وتجدر الملاحظة أن العديد من الأكواد التي تراها على الانترنت تستخدم العامل GetType للحصول على نوع الغرض المستخدم للقفل lock object وذلك لحماية الطريقة الساكنة.

عندما تستخدم عبارات SyncLock معششة للقيام بالتزامن لأغراض مختلفة من الضروري استخدام تسلسل تعشيش متطابق أينما احتجت له في تطبيقك فالتحري عن الأقفال بالتسلسل المطابق ذاته يجنبك الوصول إلى حالة الأقفال الميتة خلال العديد من أجزاء التطبيق وهذه القاعدة تنطبق أيضا عندما تقوم دالة تحتوي على SyncLock باستدعاء دالة أخرى تحتوي على SyncLock

```
' Always use this sequence when locking objLock1 and objLock2.
SyncLock objLock1
   SyncLock objLock2
      …
   End SyncLock
End SyncLock
```

اعتبارات الأداء والتواجد الكسول Performance Considerations and Lazy Instantiation

تضمين جميع الأكواد التي تستخدم متغيرات مشتركة ضمن كتلة SyncLock يؤدي إلى إبطاء تطبيقك كثيرا أو تخفيض أداؤه بشكل ملحوظ وبشكل خاص عندما يتم تشغيله على حاسب متعدد المعالجات فإن استطعت تجنب استخدام كتلة SyncLock بدون تعريض تكامل البيانات للخطر يجب عليك القيام به قطعيا فمثلا تخيل أنك تستخدم نمط وحيد بتواجد كسول lazy instantiation في بيئة متعددة المسارات

```
Public Class Singleton
   Private Shared m_Instance As Singleton
   Private Shared sharedLock As New Object()

   Public Shared ReadOnly Property Instance() As Singleton
      Get
         SyncLock sharedLock
            If m_Instance Is Nothing Then m_Instance = New Singleton
            Return m_Instance
         End SyncLock
      End Get
   End Property
End Class
```

تكمن المشكلة في الكود السابق أن معظم الوصولات للخاصية لا يحتاج إلى تزامن وذلك لأن المتغير الخاص m_Instance يتم تعيينه مرة واحدة في المرة الأولى التي يتم فيها قراءة الخاصية وفي ما يلي طريقة أفضل لتحقيق التصرف المطلوب

```
Class Singleton
   Private Shared m_Instance As Singleton
   Private Shared sharedLock As New Object

   Public Shared ReadOnly Property Instance() As Singleton
      Get
         If m_Instance Is Nothing Then
            SyncLock sharedLock
               If m_Instance Is Nothing Then m_Instance = New Singleton()
            End SyncLock
         End If
         Return m_Instance
      End Get
   End Property
End Class
```

الأغراض المتزامنة Synchronized Objects

مشكلة أخرى متعلقة بالمسارات في الدوت نيت هي أن ليس جميع أغراض الدوت نيت .NET object قابلة للمشاركة بأمان عبر المسارات not all .NET objects are thread-safe فعندما تقوم بكتابة تطبيق متعدد المسارات يجب عليك التأكد دوما من الوثائق للتأكد من أن الأغراض والطرائق التي تستخدمها آمنة للاستخدام عبر المسارات فعلى سبيل المثال جميع الطرق الساكنة للفئات Regex و Match أمنة عبر المسارات ولكن الطرق الغير ساكنة غير آمنة فيجب عدم استخدامها ضمن مسار مختلف وكذلك بعض أغراض الدوت نيت مثل Windows Forms objects and controls لها العديد من الحدود التي تجعل فقط المسار الذي أنشأها يمكنه استدعاء طرقها وخصائصها

أنواع دوت نيت المتزامنة Synchronized .NET Types

العديد من الأغراض الغير آمنة عبر المسارات بطبيعتها مثل ArrayList و Hashtable و Queue و SortedList و Stack و TextReaderو TextWriter و التعابير النظامية تقدم طريقة ساكنة قابلة للتزامن تعيد غرض أمن للمسارات thread-safe object مكافئ للذي تم تمريره كما أن معظمها يعرض الخاصية IsSynchronized التي تعيد True عندما تتعامل مع نسخة آمنة عبر المسارات

```
' Create an ArrayList object, and add some values to it.
Dim al As New ArrayList()
al.Add(1): al.Add(2): al.Add(3)
' Create a synchronized, thread-safe version of this ArrayList.
Dim syncAl As ArrayList = ArrayList.Synchronized(al)
' Prove that the new object is thread-safe.
Console.WriteLine(al.IsSynchronized)         ' => False
Console.WriteLine(syncAl.IsSynchronized)     ' => True
' You can now share the syncAl object among different threads
```

تذكر دائما أن التعامل مع هذه النسخة المتزامنة يكون أبطأ من النسخة الغير متزامنة وذلك بسبب أن كل طريقة تمر عبر سلسلة من الفحوصات الداخلية وفي معظم الحالات يمكنك كتابة كود فعال أكثر إذا استخدمت المصفوفات والمجموعات العادية regular arrays and collections وقمت بمزامنة عناصرها باستخدام كتلة SyncLock العادية

The Synchronization Attribute

استخدام الخاصية System.Runtime.Remoting.Contexts.Synchronization هي أبسط طريقة لتحقيق الوصول المتزامن للغرض Object بأكمله وبذلك يستطيع مسار واحد فقط الوصول إلى حقوله وطرائقه وبذلك أي مسار يستطيع استخدام الفئة ولكن مسار واحد فقط يستطيع تنفيذ أحد طرائقه إذا كانت الطريقة تنفذ كودا ضمن الفئة Class وأي مسار يحاول استخدام هذه الفئة عليه الانتظار وبكلمات أخرى وكأن هناك كتل SyncLock تغلف كافة طرائق الفئة مستخدمة نفس متغير الإقفال. والكود التالي يبين كيف يمكنك مزامنة فئة باستخدام الخاصية Synchronization attribute لاحظ أيضا أن الفئة يجب أن يتم وراثتها من ContextBoundObject ليتم تعليمها كـ context-bound object

```
System.Runtime.Remoting.Contexts.Synchronization()> _
Class Display
   Inherits ContextBoundObject
   …
End Class
```

و خاصية التزامن Synchronization attribute تضمن الوصول المتزامن لجميع الحقول والخصائص والطرق ولكنها لا توفر التزامن للأعضاء الساكنين static members وهي تأخذ محددا اختياريا يمكن أن تكون قيمته True أو False أو أحد الثوابت التي توفرها الفئة SynchronizationAttribute والتي يمكنك الاطلاع عليها من مكتبة MSDN

The MethodImpl Attribute

في معظم الحالات مزامنة فئة كاملة ستقتل التطبيق وحماية بعض الطرائق في تلك الفئة يكون كافيا في معظم الحالات حيث يمكنك تطبيق هذا بتغليف كود الطريقة بكتلة SyncLock أو يمكنك استخدام تقنية أبسط مبنية على الصفة System.Runtime.CompilerServices.MethodImpl

```
Class MethodImplDemoClass
    ' This method can be executed by one thread at a time.
    <MethodImpl(MethodImplOptions.Synchronized)> _
    Sub SynchronizedMethod()
        …
    End Sub
End Class
```

فتطبيق الصفة MethodImpl على عدة طرائق في الفئة يؤدي نفس الغرض من تغليف كامل تلك الطرائق بكتلة SyncLock والتي تستخدم Me كمتغير إقفال وبكلمات أخرى أي مسار يستدعي طريقة معلمة بالخاصية MethodImpl سوف يمنع أي مسار آخر من استدعاء الطريقة المعلمة بالخاصية MethodImpl كما يمكنك استخدام هذه الخاصية على الطرائق الساكنة ويكون متغير الغرض الذي يستخدم ضمنيا لقفل الطرائق الساكنة مختلف عن متغير الغرض المستخدم لقفل الطرائق الأخرى للفئة instance methods وبهذا فالمسار الذي يستدعي طريقة ساكنة معلمة بالصفة MethodImpl لا يمنع مسار آخر من استدعاء الطرائق الغير ساكنة instance methods والمعلمة بنفس الصفة

عمليات القراءة والكتابة المتغيرة Volatile Read and Write Operations

عندما تتم مشاركة متغير عبر عدة مسارات والتطبيق يعمل على حاسب متعدد المعالجات يجب عليك وضع احتمال حدوث أخطاء إضافية في الحسبان وتكمن المشكلة في النظام متعدد المعالجات في أن لكل معالج الكاش الخاص به ولهذا فإذا قمت بالكتابة على حقل في فئة على مسار سيتم كتابة القيمة الجديدة في الكاش المرتبط مع المعالج الحالي ولا يتم نشرها مباشرة إلى الكاش الخاص ببقية المعالجات بحيث يمكنهم جميعا رؤية القيمة الجديدة. كما تحدث مشكلة مشابهة في الأنظمة ذات المعالج 64 بت الذي يمكنه إعادة ترتيب تنفيذ كتل عبارات الكود متضمنا عمليات القراءة والكتابة في الذاكرة وعملية إعادة الترتيب لم يكن لها تأثير ظاهر حتى الآن من أجل مسار واحد يستخدم جزءا معينا من الذاكرة ولكن ربما سيسبب ذلك مشكلة عندما يتم الوصول إلى نفس الجزء من الذاكرة بواسطة عدة مسارات. وتوفر الفريموورك حلان لهذه المشكلة وهما زوج من الطرائق VolatileRead و VolatileWrite والطريقة MemoryBarrier ويوفرها جميعا النوع Thread

تمكنك الطريقة VolatileWrite من كتابة متغير والتأكد من أن القيمة الجديدة يتم كتابتها آليا في الذاكرة المشتركة بين جميع المعالجات ولا تبقى في المسجل الخاص بالمعالج حيث تكون مخفية عن بقية المسارات وبالمثل تمكنك الطريقة VolatileRead من قراءة المتغير بطريقة آمنة لأنها تجبر النظام على تفريغ جميع ذواكر الكاش الموجودة قبل تنفيذ العملية وكلا الطريقتان محملتان تحميلا زائدا Overloaded بحيث تأخذ متغيرات رقمية أو غرضية Object وبالمرجع كما في قطعة الكود التالية

```
Class TestClass
    Private Shared sharedValue As Integer

    Function IncrementValue() As Integer
        Dim value As Integer = Thread.VolatileRead(sharedValue)
        value += 1
        Thread.VolatileWrite(sharedValue, value)
        Return value
    End Function
End Class
```

والطريقتان المذكورتان تعملان بشكل جيد عندما نتعامل مع المتغيرات الرقمية أو الغرضية Object ولكن لا يمكن استخدامهما من أجل أنواع أخرى من المتغيرات لأنه لا يمكنك استخدام نسخة الدالة التي تأخذ متغير من النوع Object بسبب عدم إمكانية الاعتماد على عملية التحويل عندما يكون المتغير ممررا بالمرجع مما يقودنا إلى الطريقة MemoryBarrier التي تقوم بتفريغ محتويات جميع ذواكر الكاش

الخاصة بالمعالجات إلى الذاكرة الرئيسية وبهذا تضمن لك أن جميع المتغيرات تحتوي أحدث نسخة من البيانات التي تمت كتابتها إليهم فمثلا يضمن الكود التالي أن الفئة Singleton تعمل جيدا حتى على نظام متعدد المعالجات

```
Class Singleton
    Private Shared m_Instance As Singleton
    Private Shared sharedLock As New Object()

    Public Shared ReadOnly Property Instance() As Singleton
        Get
            If m_Instance Is Nothing Then
                SyncLock sharedLock
                    If m_Instance Is Nothing Then
                        Dim tempInstance As Singleton = New Singleton()
                      ' Ensure that writes related to instantiation are flushed.
                        Thread.MemoryBarrier()
                        m_Instance = tempInstance
                    End If
                End SyncLock
            End If
            Return m_Instance
        End Get
    End Property
End Class
```

ويجب عليك استدعاء الطريقة MemoryBarrier مباشرة قبل أن يتم نشر القيمة الجديدة إلى بقية المسارات وفي المثال السابق يتم التأكد من اكتمال وضع القيمة في المتغير tempInstance قبل أن توضع في المتغير الذي ستتم مشاركته عبر المسارات

The Monitor Type

توفر كتلة SyncLock طريقة سهلة لاستخدام طريقة تتعامل مع مسائل التزامن ولكنها تكون غير ملائمة في العديد من الحالات فمثلا لا يمكن للمسار اختبار كود في كتلة SyncLock وتجنب منعه من ذلك إذا كان مسار آخر ينفذ كتلة SyncLock مرتبطة مع نفس الغرض Object وكتل SyncLock معرفة داخليا بواسطة Monitor objects التي يمكن استخدامها مباشرة للحصول على مرونة أكثر ويتم ذلك على حساب زيادة التعقيد في الكود. ولا يمكنك استخدام Monitor object وحيد وفي الحقيقة جميع طرق Monitor type التي سيتم عرضها هي طرائق ساكنة وتعتبر Enter هي الطريقة الأهم وهي تأخذ محدد من النوع Object الذي يعمل كالمحدد الممرر لكتلة SyncLock وتكون له نفس الشروط من كونه من نوع مرجعي ومشترك ولا يمكن أن يحمل القيمة Nothing وإن لم تمتلك المسارات الأخرى قفلا على هذا الغرض فيقوم المسار الحالي بطلب ذلك القفل ويضبط قيمة العداد إلى 1 وإن امتلك مسار آخر القفل يجب على المسار الطالب انتظار أن يقوم المسار الآخر بتحرير القفل حتى يصبح متوفرا وإن كان المسار الطالب يمتلك القفل أساسا يؤدي كل استدعاء للطريقة Monitor.Enter إلى زيادة قيمة العداد. وتأخذ الطريقة Monitor.Exit غرض القفل lock object كمحدد لها وتنقص قيمة العداد وعندما تصل قيمة العداد للصفر يتم تحرير القفل ممكنا بقية المسارات من الحصول عليه ويجب أن يتم الموازنة بين استدعاء الطريقة Monitor.Enter والطريقة Monitor.Exit أو لن يتم تحرير القفل أبدا

```
' A non-Nothing module-level object variable
Dim objLock As New Object()
…
Try
   ' Attempt to enter the protected section;
   ' wait if the lock is currently owned by another thread.
   Monitor.Enter(objLock)
   ' Do something here.
   …
Finally
   ' Release the lock.
   Monitor.Exit(objLock)
End Try
```

إذا كان هناك احتمال في أن تطلق العبارات الموجودة بين الطريقتان Enter و Exit استثناء يجب عليك عندها وضع كامل الكود ضمن كتلة Try...End Try لأنه من الضروري أن تقوم بتحرير القفل دوما وإن طلب مسار طريقة على مسار آخر تنتظر داخل الطريقة Monitor.Enter سوف يستقبل ذلك المسار استثناء ThreadInterruptedException الذي يعتبر سببا إضافيا لاستخدام كتلة Try...End Try والطريقتان Enter و Exit الخاصتين بـ Monitor Object يسمحان لك باستبدال كتلة SyncLock ولكنهما لا يقدمان لك أية فوائد إضافية وسوف ترى المرونة الزائدة للفئة Monitor عندما تطبق الطريقة TryEnter وهي مشابهة للطريقة Enter ولكنها تخرج وتعيد False إذا كان لا يمكن الحصول على القفل خلال فترة زمنية محددة فمثلا يمكنك محاولة الحصول على Monitor خلال 10 ميللي ثانية ثم التخلي عن ذلك دون أن توقف المسار الحالي مدة غير محددة ويقوم الكود التالي بإعادة كتابة المثال السابق المعتمد على SyncLock مستخدما Monitor object ويظهر لك المحاولات الفاشلة للحصول على القفل

```
Try
   Do Until Monitor.TryEnter(consoleLock, 10)
      Debug.WriteLine("Thread " + Thread.CurrentThread.Name + _
         " failed to acquire the lock")
   Loop
   ' Split the output line in pieces.
   Console.Write(" ")
   Console.Write(Thread.CurrentThread.Name)
Finally
   ' Release the lock.
   Monitor.Exit(consoleLock)
End Try
```

The Mutex Type

يوفر النوع Mutex مبدأ آخر للتزامن حيث أن الـ Mutex هو Windows kernel object يمكن امتلاكه من قبل مسار واحد فقط في الوقت نفسه ويكون في حالة إشارة a signaled state عندما لا يمتلكه أي مسار. ويطلب المسار ملكية الـ Mutex باستخدام الطريقة الساكنة Mutex.WaitOne والتي لا تعود إلا بعد أن يتم تحقيق الملكية ويتم تحريرها باستخدام الطريقة الساكنة Mutex.ReleaseMutex والمسار الذي يطلب ملكية Mutex object المملوك من قبله سلفا لا يمنع نفسه من الحصول على الملكية فيجب عليك في هذه الحالة استدعاء ReleaseMutex بعدد مساوي من المرات وهذا مثال عن كيفية تعريف قسم متزامن باستخدام Mutex object

```
' This Mutex object must be accessible to all threads.
Dim m As New Mutex()

Sub WaitOneExample()
   m.WaitOne()
   ' Enter the synchronized section.
   …
   ' Exit the synchronized section.
   m.ReleaseMutex()
End Sub
```

وفي التطبيقات الحقيقية عليك استخدام كتلة Try لحماية كودك من الأخطاء ووضع استدعاء ReleaseMutex في قسم Finally وإن قمت بتمرير محدد اختياري للطريقة WaitOne كزمن انتهاء فستعيد التحكم للمسار عندما يتم تحقيق الملكية بنجاح أو عندما ينتهي الوقت المحدد ويمكن معرفة الفرق بين النتيجتين باختبار القيمة المعادة حيث أن True تعني تحقيق الملكية و False تعني انتهاء الوقت

```
' Attempt to enter the synchronized section, but give up after 0.1 seconds.
If m.WaitOne(100, False) Then

   ' Enter the synchronized section.
   …
   ' Exit the synchronized section, and release the mutex.
```

```
   m.ReleaseMutex()
End If
```

عند استخدام هذه الطريقة يوفر النوع Mutex آلية مكافئة للطريقة Monitor.TryEnter بدون تقديم أية خصائص إضافية ويمكنك رؤية المرونة الإضافية للنوع Mutex عندما ترى الطريقتين الساكنتين WaitAny و WaitAll الخاصتين به والطريقة WaitAny تأخذ مصفوفة من Mutex objects وتعود عندما تحقق ملكية واحد من Mutex objects من تلك القائمة وفي هذه الحالة يصبح الـ Mutex في حالة إشارة أو عندما ينتهي الوقت المحدد بالمحدد الاختياري والقيمة المعادة تكون عبارة عن مصفوفة من Mutex objects التي أصبحت في حالة إشارة أو قيمة خاصة هي 258 عندما ينتهي الوقت المحدد. وتستخدم مصفوفة من Mutex objects عندما يكون لدينا عدد محدود من الموارد ونريد أن نربط كل واحد منها بمسار حالما يصبح ذلك المصدر متوفرا وفي هذه الحالة يصبح الـ Mutex objects الذي في حالة إشارة يعني أن المصدر الموافق متوفر عندئذ يمكنك استخدام الطريقة Mutex.WaitAny لمنع المسار الحالي حتى يصبح واحدا من الـ Mutex objects في حالة إشارة و النوع Mutex يرث الطريقة WaitAny من WaitHandle الخاصة بفئته الأب وهذا هيكل لتطبيق يستخدم هذه التقنية

```
' An array of three Mutex objects
Dim mutexes() As Mutex = {New Mutex(), New Mutex(), New Mutex()}

Sub WaitAnyExample()
   ' Wait until a resource becomes available.
   ' (Returns the index of the available resource.)
   Dim mutexNdx As Integer = Mutex.WaitAny(mutexes)
   ' Enter the synchronized section.
   ' (This code should use only the resource corresponding to mutexNdx.)
   …
   ' Exit the synchronized section, and release the resource.
   mutexes(mutexNdx).ReleaseMutex()
End Sub
```

والطريقة الساكنة WaitAll أيضا موروثة من WaitHandle الخاصة بالفئة الأب حيث تأخذ مصفوفة من Mutex objects وتعيد التحكم للتطبيق فقط عندما يصبح جميعهم في حالة إشارة وهي مفيدة بشكل خاص عندما لا يمكنك المتابعة إلا عندما تكون جميع المسارات الباقية قد أنهت عملها

```
' Wait until all resources have been released.
Mutex.WaitAll(mutexes)
```

وهناك مشكلة صغيرة متعلقة بالطريقة WaitAll هي أنه لا يمكن استدعاؤها من المسار الرئيسي في تطبيق مسار الغرفة الوحيدة Single Thread Apartment (STA) application مثل تطبيق الكونسول Console application أو تطبيق نماذج ويندوز Windows Forms application ففي المسار الرئيسي لتطبيق STA يجب عليك التوقف حتى يتم تحرير مجموعة من الـ Mutex عندها يجب عليك استخدام WaitAll من مسار منفصل ثم استخدام الطريقة Thread.Join على ذلك المسار لإيقاف المسار الرئيسي حتى تعود الطريقة WaitAll وفي فيجول بايزيك 2005 والنسخة 2 من الفريموورك يوجد الطريقة الساكنة الجديدة SignalAndWait تمكنك من وضع Mutex object في حالة إشارة وانتظار Mutex object آخر

```
' Signal the first mutex and wait for the second mutex to become signaled.
Mutex.SignalAndWait(mutexes(0), mutexes(1))
```

وخلافا لجميع أغراض التزامن التي تم ذكرها حتى الآن يمكن لـ Mutex objects أن يرتبط باسم الأمر الذي يعد من أهم المزايا لهذه الأغراض فأغراض Mutex objects التي تمتلك نفس الاسم يمكن مشاركتها عبر العمليات ويمكنك إنشاء تواجد Instance لها كما يلي

```
Dim m As New Mutex(False, "mutexname")
```

وإن كان الاسم موجودا سابقا في النظام يحصل المستدعي على مرجع له وإلا سيتم إنشاء Mutex object جديد بحيث تمكنك هذه الآلية من مشاركة Mutex objects عبر عدة تطبيقات مختلفة وبهذا تتمكن هذه التطبيقات من مزامنة عمليات الوصول للمصادر المختلفة وقد تم إضافة باني جديد في الفريموورك 2 وفيجول بايزيك 2005 يمكنك من اختبار إذا كان قد تم منح المسار المستدعي ملكية الـ Mutex

```
Dim ownership As Boolean
Dim m As New Mutex(True, "mutexname", ownership)
If ownership Then
   ' This thread owns the mutex.
   …
End If
```

من الاستخدامات الشائعة لـ named mutexes هو تحديد فيما إذا كان التطبيق العامل هو الأول أو الوحيد الذي تم تحميله وإن لم تكن هذه الحالة يمكن للتطبيق الخروج مباشرة أو الانتظار حتى تنتهي النسخة الأخرى من مهامها كما في المثال

```
Sub Main()
   Dim ownership As Boolean
   Dim m As New Mutex(True, "DemoMutex", ownership)
   If ownership Then
      Console.WriteLine("This app got the ownership of Mutex named DemoMutex")
      Console.WriteLine("Press ENTER to run another instance of this app")
      Console.ReadLine()
      Process.Start(Assembly.GetExecutingAssembly().GetName().CodeBase)
   Else
      Console.WriteLine("This app is waiting to get ownership of Mutex named
DemoMutex")
      m.WaitOne()
   End If
   ' Perform the task here.
   …
   Console.WriteLine("Press ENTER to release ownership of the mutex")
   Console.ReadLine()
   m.ReleaseMutex()
End Sub
```

والطريقة الساكنة OpenExisting جديدة أيضا في الفريموورك 2 وتقدم طريقة أخرى لفتح Mutex على مستوى النظام named system-wide Mutex object وبعكس باني الـ Mutex تمكنك هذه الطريقة من تحديد درجة التحكم التي تريدها على الـ Mutex

```
Try
   ' Request a mutex with the right to wait for it and to release it.
   Dim rights As MutexRights = MutexRights.Synchronize Or MutexRights.Modify
   Dim m As Mutex = Mutex.OpenExisting("mutexname", rights)
   ' Use the mutex here.
   …
Catch ex As WaitHandleCannotBeOpenedException
   ' The specified object doesn't exist.
Catch ex As UnauthorizedAccessException
   ' The specified object exists, but current user doesn't have the
   ' necessary access rights.
Catch ex As IOException
   ' A Win32 error has occurred.
End Try
```

وفي فيجول بايزيك 2005 والفريموورك 2 تظهر الميزة الجديدة الأهم في النوع Mutex وهي إمكانية الوصول لقوائم التحكم بالوصول access control lists (ACLs) في النموذج عبر الغرض System.Security.AccessControl.MutexSecurity object حيث

يمكنك تحديد ACL عندما تنشئ غرض Mutex جديد مستخدما الطريقة GetAccessControl للحصول على غرض MutexSecurity المرتبط بـ Mutex محدد وتطبيق ACL جديد باستخدام الطريقة SetAccessControl

```
Dim ownership As Boolean
Dim m As New Mutex(True, "mutexname", ownership)
If Not ownership Then
   ' Determine who is the owner of the mutex.
   Dim mutexSec As MutexSecurity = m.GetAccessControl()
   Dim account As NTAccount = DirectCast(mutexSec.GetOwner( _
   GetType(NTAccount)), NTAccount)
   Console.WriteLine("Mutex is owned by {0}", account)
End If
```

The Semaphore Type

تقدم الفريموورك 2 و فيجول بايزيك دوت نيت نوعا جديدا وهو Semaphore type الذي يرتكز على Win32 semaphore object وخلافا لبقية أغراض المسارات الموجودة في المكتبة mscorlib فهذا النوع تم تعريفه في المكتبة system.dll وهو يستخدم عندما تريد تحديد حد أقصى (عدد N) من المسارات التي يمكن تنفيذها في جزء معين من الكود أو للوصول إلى مصدر معين ويمتلك عددا ابتدائيا وعددا أقصى ويجب عليك تمرير هذه القيم لبانيه

```
' A semaphore that has an initial count of 1 and a maximum count of 2.
Dim sem As New Semaphore(1, 2)
```

يحاول المسار أخذ ملكية الـ semaphore باستدعاء الطريقة WaitOne وإن كان العدد الحالي أكبر من الصفر يتم إنقاصه وتعود الطريقة مباشرة وإلا تنتظر حتى يحرر مسار آخر semaphore أو إنقضاء الوقت المحدد بالمحدد الاختياري ويحرر المسار semaphore باستدعاء الطريقة Release مما يزيد العدد بمقدار 1 أو بقيمة محددة ويعيد قيمة العدد السابق

```
Dim sem As New Semaphore(2, 2)
' Next statement brings count from 2 to 1.
sem.WaitOne()
…
' Next statement brings count from 1 to 2.
sem.Release()
' Next statement attempts to bring count from 2 to 3, but
' throws a SemaphoreFullException.
sem.Release()
```

وبشكل أساسي ستستخدم الغرض Semaphore كما يلي

```
' Initial count is initially equal to max count.
Dim sem2 As New Semaphore(2, 2)

Sub Semaphore_Example()
   ' Wait until a resource becomes available.
   sem2.WaitOne()
   ' Enter the synchronized section.
   …
   ' Exit the synchronized section, and release the resource.
   sem2.Release()
End Sub
```

تذكر دوما استخدام الكتلة Try…Finally للتأكد من أن الـ semaphore قد تم تحريره حتى لو حدث استثناء ما وتماما كالـ mutexes يمكن للـ semaphores امتلاك اسم ومشاركته عبر العمليات وعندما تحاول إنشاء غرض semaphores موجود سابقا يتم تجاهل العدد والحد الأقصى

```
Dim ownership As Boolean
Dim sem3 As New Semaphore(2, 2, "semaphoreName", ownership)
If ownership Then
   ' Current thread has the ownership of the semaphore.
   …
End If
```

ويدعم الغرض Semaphore أيضا الـ ACLs التي يمكن تمريرها للباني حيث تتم القراءة بواسطة الطريقة GetAccessControl والتعديل بواسطة الطريقة SetAccessControl ومن الضروري أن تلاحظ أن النوع Mutex والنوع Semaphore تتم وراثتهما من الفئة الأساسية WaitHandle لذا يمكن تمريرهما كمحددات للطرائق الساكنة WaitAny و WaitAll و SignalAndWait للنوع WaitHandle مما يمكنك من مزامنة المصادر بسهولة والتي تكون محمية بواسطة أيا من هذه الأغراض كما في الكود

```
' Wait until two mutexes, two semaphores, and one event object become signaled.
Dim waitHandles() As WaitHandle = {mutex1, mutex2, sem1, sem2, event1}
WaitHandle.WaitAll(waitHandles)
```

The ReaderWriterLock Type

العديد من المصادر في العالم الحقيقي يمكن إما القراءة منها أو الكتابة إليها وهي تدعم في الغالب إما مجموعة قراءات متعددة أو عملية كتابة وحيدة يتم تنفيذها في لحظة معينة فمثلا يمكن لعدة عملاء القراءة من ملف بيانات أو جدول في قاعدة بيانات ولكن إن تمت الكتابة للملف أو الجدول فلا يمكن حدوث أي عمليات قراءة أو كتابة على ذلك المصدر حيث يمكنك تعريف قفلا لكتابة واحدة أو عدة قراءات بدلالة الغرض ReaderWriterLock واستخدام هذا الغرض يعتبر رؤية إلى الأمام فكل المسارات التي تريد استخدام مصدر معين يجب عليها استخدام نفس الغرض ReaderWriterLock وقبل محاولة القيام بأي عملية على ذلك المصدر يجب على المسار استدعاء إما الطريقة AcquireReaderLock أو الطريقة AcquireWriterLock وذلك اعتمادا على العملية التي يتم تنفيذها وهذه الطرائق تقوم بإيقاف المسار الحالي حتى يتم الحصول على ذلك المصدر وأخيرا على المسار استدعاء الطريقة ReleaseReaderLock أو الطريقة ReleaseWriterLock عندما تنهي عملية القراءة أو الكتابة على ذلك المصدر والمثال التالي يقوم بإنشاء 10 مسارات تقوم بعملية قراءة أو كتابة على مصدر مشترك

```
Dim rwl As New ReaderWriterLock()
Dim rnd As New Random()

Sub TestReaderWriterLock()
   For i As Integer = 0 To 9
      Dim t As New Thread(AddressOf ReaderWriterLock_Task)
      t.Start(i)
   Next
   …
End Sub

Sub ReaderWriterLock_Task(ByVal obj As Object)
   Dim n As Integer = CInt(obj)
   ' Perform 10 read or write operations. (Reads are more frequent.)
   For i As Integer = 1 To 10
      If rnd.NextDouble < 0.8 Then
         ' Attempt a read operation.
         rwl.AcquireReaderLock(Timeout.Infinite)
         Console.WriteLine("Thread #{0} is reading", n)
         Thread.Sleep(300)
         Console.WriteLine("Thread #{0} completed the read operation", n)
         rwl.ReleaseReaderLock()
      Else
```

```
        ' Attempt a write operation.
        rwl.AcquireWriterLock(Timeout.Infinite)
        Console.WriteLine("Thread #{0} is writing", n)
        Thread.Sleep(300)
        Console.WriteLine("Thread #{0} completed the write operation", n)
        rwl.ReleaseWriterLock()
      End If
   Next
End Sub
```

وعندما تشغل هذا الكود سترى أن عدة مسارات يمكنها القراءة بنفس الوقت والمسار الذي يقوم بالكتابة يوقف جميع المسارات الأخرى ويمكن للطريقتان AcquireReaderLock و AcquireWriterLock أخذ محدد عبارة عن زمن انتهاء timeout وذلك بقيمة من النوع TimeSpan أو بعدد من الميللي ثانية ويمكنك اختبار فيما إذا تم الحصول على القفل بنجاح باستخدام الخصائص IsReaderLockHeld و IsWriterLockHeld القابلة للقراءة فقط إذا مررت قيمة غير Timeout.Infinite

```
' Attempt to acquire a reader lock for no longer than 1 second.
rwl.AcquireWriterLock(1000)
If rwl.IsWriterLockHeld Then
   ' The thread has a writer lock on the resource.
   …
End If
```

والمسار الذي يمتلك قفل القراءة يمكنه الترقية إلى قفل للكتابة باستدعاء الطريقة UpgradeToWriterLock والعودة ثانية لوضع القراءة باستخدام الطريقة Downgrade-FromWriterLock والشئ الرائع بخصوص الأغراض ReaderWriterLock هي أنها أغراض خفيفة بحيث يمكن استخدامها عددا كبيرا من المرات دون أن تؤثر على الأداء بشكل ملحوظ وبما أن الطرائق AcquireReaderLock و AcquireWriterLock تأخذان وقت انتهاء فالتطبيق المصمم بشكل جيد يجب أن لا يعاني من أقفال ميتة ومع ذلك يمكن حصول حالة قفل ميت عندما يكون مساران ينتظران مصدرا محجوزا من قبل مسار لا يقوم بتحريره حتى انتهاء العملية الجارية

The Interlocked Type

يزودنا النوع Interlocked بطريقة للقيام بعمليات دقيقة لزيادة أو إنقاص قيمة متغير مشترك وهذه الفئة تعرض فقط طرائق ساكنة (لا نحتسب هنا ما تمت وراثته من Object) انظر إلى الكود التالي

```
' Increment and Decrement methods work with 32-bit and 64-bit integers.
Dim lockCounter As Integer
…
' Increment the counter and execute some code if its previous value was zero.
If Interlocked.Increment(lockCounter) = 1 Then
   …
End If
' Decrement the shared counter.
Interlocked.Decrement(lockCounter)
```

والطريقة ADD جديدة في الفريموورك 2 وهي تمكنك من زيادة أعداد حقيقية Integer من عيار 32 أو 64 بت بقيمة محددة

```
If Interlocked.Add(lockCounter, 2) <= 10 Then…
```

وتوفر الفئة Interlocked طريقتان ساكنتان أخريان الطريقة Exchange التي تمكنك من تحديد قيمة من اختيارك إلى متغيرات من النوع Integer أو Long أو Single أو Double أو IntPtr أو Object وتعيد القيمة السابقة وبما أن لها نسخة محملة زائدا تأخذ محددا من النوع Object لهذا يمكنك أن تجعلها تعمل لأي نوع مرجعي كالنوع String كما في المثال

```
Dim s1 As String = "123"
Dim s2 As String = Interlocked.Exchange(s1, "abc")
Console.WriteLine("s1={0}, s2={1}", s1, s2)
```

والطريقة CompareExchange تعمل بأسلوب مشابهة ولكنها تقوم بالتبديل فقط إذا كان موقع الذاكرة مساوي لقيمة محددة يتم تمريرها لها

The ManualResetEvent, AutoResetEvent, and EventWaitHandle Types

هذه الفئات الثلاثة تعمل بشكل متشابه ManualResetEvent و AutoResetEvent و EventWaitHandle والفئة الأخيرة هي الفئة الأب للفئتان الأولان وقد تمت إضافتها في الفريموورك 2 على الرغم من أن ManualResetEvent و AutoResetEvent لم يتم إهمالهما بعد وأثناء العمل يمكنك استبدالهما بالفئة الجديدة EventWaitHandle التي تعطيك مزيدا من المرونة عند التعامل. والنوعان ManualResetEvent و AutoResetEvent مفيدان بشكل خاص عندما تريد إيقاف مسار أو أكثر بشكل مؤقت حتى يخبرنا مسار آخر بأنه لا مانع من المتابعة وتستخدمهما لإيقاظ مسار مثل إجراء معالجة الحدث في مسار متوقف ولكن لا تنخدع بوجود Event في أسمائهما فلا يمكنك استخدام إجراءات معالجة الحدث التقليدية مع هذه الأغراض. وكائن من أحد هذين النوعين يمكن أن يكون في حالة إشارة أو عدم إشارة Signale/UnSignaled وهذه القيمة لا تملك أي معنى خاص بحيث يمكنك اعتبارها كحالة تشغيل/إيقاف حيث ستمرر الحالة الابتدائية للباني وأي مسار يستطيع الوصول لذلك الغرض يمكنه ضبط تلك الحالة إلى Signaled باستخدام الطريقة Set أو يستخدم الطريقة Reset لإعادة الحالة إلى UnSignaled ويمكن للمسارات الأخرى استخدام الطريقة WaitOne للانتظار حتى تصبح في حالة إشارة Signaled أو حتى انتهاء فترة الانتظار

```
' Create an auto reset event object in nonsignaled state.
Dim are As New AutoResetEvent(False)
' Create a manual reset event object in signaled state.
Dim mre As New ManualResetEvent(True)
```

والاختلاف الوحيد بين الغرضان ManualResetEvent و AutoResetEvent هو أن الأخير يعيد ضبط نفسه آليا (يصبح في حالة عدم إشارة Unsignaled) وذلك مباشرة بعد أن يتم صد المسار عندما تبدأ الطريقة WaitOne ويوقظ الغرض AutoResetEvent فقط واحد من المسارات المنتظرة عندما يصبح في حالة إشارة بينما الغرض ManualResetEvent يوقظ جميع المسارات المنتظرة ويجب أن يتم إعادة ضبطه يدويا إلى حالة عدم إشارة كما هو ظاهر من اسمه وكما ذكر سابقا يمكنك استبدال الغرضين ManualResetEvent و AutoResetEvent بالغرض EventWaitHandle كما يظهر بالكود التالي

```
' These statements are equivalent to the previous code example.
Dim are As New EventWaitHandle(False, EventResetMode.AutoReset)
Dim mre As New EventWaitHandle(True, EventResetMode.ManualReset)
```

وتكون أغراض الـ Event مفيدة خاصة في حالات المنتج والمستهلك فربما يكون لديك إجراء وحيد في مسار يقوم بتقييم بعض البيانات أو بالقراءة من القرص أو منفذ تسلسلي أو غيرها ويستدعي الطريقة Set على غرض متزامن فيتم إعادة تشغيل مسار أو أكثر لمعالجة تلك البيانات ويجب عليك استخدام الغرض AutoResetEvent أو الغرض EventWaitHandle مع الخيار AutoReset إذا كان هناك مسار مستهلك وحيد سيقوم بمعالجة تلك البيانات كما يجب عليك استخدام الغرض ManualResetEvent أو الغرض EventWaitHandle مع الخيار ManualReset إذا كان يجب معالجة البيانات باستخدام جميع المسارات المستهلكة.

ويبين المثال التالي كيف يمكن أن يكون لديك عدة مسارات منتجة تقوم بعملية البحث عن ملف في عدة مجلدات مختلفة في نفس الوقت ولكن يوجد مسار مستهلك وحيد يقوم بجمع النتائج من تلك المسارات ويستخدم المثال الغرض AutoResetEvent لإيقاظ المسار المستهلك عندما يتم إضافة اسم ملف جديد للقائمة List(Of String) ويستخدم أيضا الفئة Interlocked لإدارة عدد المسارات العاملة حتى يعلم المسار الرئيسي أنه لم تعد توجد أي بيانات أخرى لاستهلاكها

```
' The shared AutoResetEvent object
Public are As New AutoResetEvent(False)
```

```
' The list where matching filenames should be added
Public fileList As New List(Of String)()
' The number of running threads
Public searchingThreads As Integer
' An object used for locking purposes
Public lockObj As New Object()

Sub TestAutoResetEvent()
   ' Search *.zip files in all the subdirectories of C.
   For Each dirname As String In Directory.GetDirectories("C:\")
      Interlocked.Increment(searchingThreads)
      ' Create a new wrapper class, pointing to a subdirectory.
      Dim sf As New FileFinder()
      sf.StartPath = dirname
      sf.SearchPattern = "*.zip"
      ' Create and run a new thread for that subdirectory only.
      Dim t As New Thread(AddressOf sf.StartSearch)
      t.Start()
   Next

   ' Remember how many results we have so far.

   Dim resCount As Integer = 0
   Do While searchingThreads > 0
      ' Wait until there are new results.
      are.WaitOne()

      SyncLock lockObj
         ' Display all new results.
         For i As Integer = resCount To fileList.Count - 1
            Console.WriteLine(fileList(i))
         Next
         ' Remember that you've displayed these filenames.
         resCount = fileList.Count
      End SyncLock
   Loop
   Console.WriteLine("")
   Console.WriteLine("Found {0} files", resCount)
End Sub
```

وكل مسار إجرائي يعمل ضمن غرض FileFinder مختلف الذي يجب أن يكون قادرا على الوصول إلى متغيرات عامة محددة في الكود السابق

```
Class FileFinder
   Public StartPath As String       ' The starting search path
   Public SearchPattern As String   ' The search pattern

   Sub StartSearch()
      Search(Me.StartPath)
      ' Decrease the number of running threads before exiting.
      Interlocked.Decrement(searchingThreads)
      ' Let the consumer know it should check the thread counter.
      are.Set()
   End Sub

   ' This recursive procedure does the actual job.
   Sub Search(ByVal path As String)
      ' Get all the files that match the search pattern.
      Dim files() As String = Directory.GetFiles(path, SearchPattern)
      ' If there is at least one file, let the main thread know about it.
      If files IsNot Nothing AndAlso files.Length > 0 Then
```

```
         ' Ensure found files are added as an atomic operation.
         SyncLock lockObj
            ' Add all found files.
            fileList.AddRange(files)
            ' Let the consumer thread know about the new filenames.
            are.Set()
         End SyncLock
      End If

      ' Repeat the search on all subdirectories.
      For Each dirname As String In Directory.GetDirectories(path)
         Search(dirname)
      Next
   End Sub
End Class
```

والنسخة 2 من الفريموورك وفيجول بايزيك 2005 تستخدم EventWaitHandle بدلا عن AutoResetEvent أو ManualResetEvent مما يعطيك ميزة هامة وهي إمكانية إنشاء غرض مسمى على مستوى النظام يمكن مشاركته مع العمليات الأخرى والصيغة العامة لباني EventWaitHandle مشابهة لتلك الخاصة بالفئة Mutex

```
Create a system-wide auto reset event that is initially in the signaled state.
Dim ownership As Boolean
Dim ewh As New EventWaitHandle(True, EventResetMode.AutoReset, "eventname",
ownership)
If ownership Then
   ' The event object was created by the current thread.
   …
End If
```

كما يمكنك استخدام الطريقة OpenExisting لفتح غرض حدث موجود

```
' This statement throws a WaitHandleCannotBeOpenedException if the specified
' event doesn't exist, or an UnauthorizedAccessException if the current
' user doesn't have the required permissions.
ewh = EventWaitHandle.OpenExisting("eventname", _
               EventWaitHandleRights.FullControl)
```

والميزة الأخرى الهامة في أغراض الأحداث event objects في الفريموورك 2 هي دعم ACLs باستخدام الطرائق SetAccessControl و GetAccessControl والتي تأخذ وتعيد كائن من النوع EventWaitHandleSecurity حيث يمكنك استخدامه بنفس طريقة استخدام مثيلاتها في الغرض Mutex وأغراض الدوت نيت الأخرى التي تدعم ACLs

كيفية تنفيذ عملية في مسار آخر وإظهار النتيجة في التحكمات على النموذج

سألني أحد الإخوة عن مشكلة واجهته عند تنفيذ عملية معينة على مسار آخر ومحاولته إظهار النتيجة على النموذج فإذا افترضنا أنه لدينا إجراء بسيطا ينفذ عملية ما ونفذناه على مسار آخر Thread غير المسار الرئيسي الذي تنفذ عليه عمليات البرنامج وأن ذلك الإجراء يحتوي على كود يقوم بضبط قيمة الخاصية Text لصندوق نصوص على النموذج فعند تنفيذ الكود ستحصل على رسالة خطأ

Cross-thread operation not valid: Control 'TextBox1' accessed from a thread other than the thread it was created on.

وإذا أردت توليد رسالة الخطأ السابق بنفسك أنشئ مشروعا جديدا ضع عليه صندوق نصوص وزر واجعل كود النموذج مطابقا لما يلي ثم قم بتشغيل البرنامج وستحصل على رسالة الخطأ السابقة

```
Imports System.Threading

Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        Dim th As New Thread(AddressOf DoLongOperation)
        th.Start()

    End Sub

    Private Sub DoLongOperation()
        Me.TextBox1.Text = "Something"
    End Sub

End Class
```

الحل الذي أقوم باستخدامه عادة لحل هكذا مشكلة هو إنشاء فئة Class تقوم بتنفيذ العملية على المسار الثاني وتعيد النتيجة للنموذج من خلال إطلاق حدث يعيد القيم الناتجة عن عملية المعالجة للنموذج فربما لا تكون هذه هي الطريقة الأفضل في جميع الحالات ولكنها طريقتي على كل حال وسأقوم بشرحها ثم يمكننا النقاش وتجربة أية حلول أخرى لتجاوز هذه المشكلة وسأطرحها عبر تنفيذ عداد بسيط للوقت فربما ستستخدم أنت هذه الطريقة للبحث عن ملفات أو تنفيذ عمليات معالجة معقدة تستغرق وقتا طويلا ولكنني هنا اخترت مثالا يعيد قيمة وحيدة بحيث يكون بسيطا قدر الإمكان

الآن سأقوم بإضافة فئة Class جديد للمشروع يتم عبره تنفيذ العملية الطويلة التي نريد تنفيذها على مسار آخر وسأقوم بتسميتها MyStopWatch في الوقت الحالي وبما أننا سنتعامل مع المسارات سنحتاج للاستيراد التالي قبل تعريف الفئة

```
Imports System.Threading
```

سأقوم بتعريف فئة فرعية داخل الفئة MyStopWatch باسم ReturnValueEventArgs سأستخدمها لاحقا لإطلاق الحدث الذي سيعيد النتيجة إلى النموذج وهذه يجب أن تكون موروثة من الفئة EventArgs بما أنها فئة خاصة بإعادة قيم الحدث الذي سيتم إطلاقه وسأعرف فيها خاصية وحيدة ReturnVlaue ستكون للقراءة فقط بما أننا لن نحتاج لضبط قيمتها إلا من خلال باني الفئة تعيد القيمة وباني للفئة يمرر له قيمة نصية وحيدة تمثل القيمة المعادة وبهذا يكون كود الفئة ReturnValueEventArgs كما يلي

```
Public Class ReturnValueEventArgs
    Inherits EventArgs

    Private _ReturnValue As String

    Public ReadOnly Property ReturnVlaue() As String
        Get
            Return _ReturnValue
        End Get
    End Property
```

```
    Public Sub New(ByVal RetVal As String)
        _ReturnValue = RetVal
    End Sub

End Class
```

ضمن كود الفئة MyStopWtach وبعد نهاية تعريف الفئة ReturnValueEventArgs نقوم بتعريف الحدث الذي سنقوم بإطلاقه ليعيد القيمة إلى النموذج ومن أجل الالتزام بتنسيق الأحداث كما نرى في التحكمات والفئات قمنا بتعريف الفئة ReturnValueEventArgs وبهذا يكون تعريف الحدث في قسم تعريف المتغيرات العامة في الفئة MyStopWatch كما يلي

```
Public Event ReturnValue(ByVal sender As Object, ByVal e As ReturnValueEventArgs)
```

عرف متغيرا عاما على مستوى الفئة MyStopWtach باسم _MyTimer وهو من النوع Stopwatch كما يلي

```
Private _MyTimer As Stopwatch
```

حيث سنستخدمه كعداد للوقت من أجل الحصول على قيمة ليتم إعادتها ضمن إجراء المعالجة الذي سيتم تنفيذه على المسار الآخر بحيث سيكون كود إجراء المعالجة الذي سينفذ على المسار الثاني كما يلي

```
Private Sub DoProcessing()
    Do
        Dim Ret = _MyTimer.Elapsed.Hours & ":" & _
                  _MyTimer.Elapsed.Minutes & ":" & _
                  _MyTimer.Elapsed.Seconds & ":" & _
                  _MyTimer.Elapsed.Milliseconds

        RaiseEvent ReturnValue(Me, New ReturnValueEventArgs(Ret))
    Loop Until _MyTimer.IsRunning = False
End Sub
```

حيث وضعنا قيمة العداد في متغير نصي Ret ثم استخدمنا الدالة RaiseEvent لإطلاق الحدث ReturnValue حيث القيمة Me التي تشير للفئة الحالية كبارمتر أول للحدث ReturnValue يكون المحدد الثاني للحدث عبارة عن كيان Instance من الفئة ReturnValueEventArgs التي نمرر لبانيها المتغير Ret الذي يشكل القيمة المعادة من الخاصية ReturnVlaue العائدة للفئة ReturnValueEventArgs عندما سنستقبلها من النموذج

وسيكون لدينا إجراء لبدء تنفيذ المؤقت على المسار الثاني باسم StartTimer بحيث يكون كوده على الشكل

```
Public Sub StartTimer()
    _MyTimer.Reset()
    _MyTimer.Start()

    Dim th As New Thread(AddressOf DoProcessing)
    th.Start()
End Sub
```

حيث قمنا بتصفير العداد وبدئه ثم عرفنا مسارا جديدا th يقوم بتنفيذ الإجراء DoProcessing ومن أجل إيقاف العداد سنحتاج لإجراء StopTimer يكون كوده على الشكل

```
Public Sub StopTimer()
    _MyTimer.Stop()
End Sub
```

وبهذا تكون قد اكتملت فئتنا التي ستقوم بعملية المعالجة على مسار ثاني وتعيد قيمة نصية سنقوم بعرضها في صندوق نصوص لاحقا ويكون بذلك الكود الكامل لهذه الفئة

```
Imports System.Threading

Public Class MyStopWtach

    Public Class ReturnValueEventArgs
        Inherits EventArgs

        Private _ReturnValue As String

        Public ReadOnly Property ReturnVlaue() As String
            Get
                Return _ReturnValue
            End Get
        End Property

        Public Sub New(ByVal RetVal As String)
            _ReturnValue = RetVal
        End Sub

    End Class

    Public Event ReturnValue(ByVal sender As Object, _
                ByVal e As ReturnValueEventArgs)

    Private _MyTimer As New Stopwatch

    Public Sub StartTimer()
        _MyTimer.Reset()
        _MyTimer.Start()

        Dim th As New Thread(AddressOf DoProcessing)
        th.Start()
    End Sub

    Private Sub DoProcessing()
        Do
            Dim Ret = _MyTimer.Elapsed.Hours & ":" & _
                      _MyTimer.Elapsed.Minutes & ":" & _
                      _MyTimer.Elapsed.Seconds & ":" & _
                      _MyTimer.Elapsed.Milliseconds

            RaiseEvent ReturnValue(Me, New ReturnValueEventArgs(Ret))
        Loop Until _MyTimer.IsRunning = False
    End Sub

    Public Sub StopTimer()
        _MyTimer.Stop()
    End Sub

End Class
```

نعود للنموذج الخاص بالمشروع الذي نحتاج لوجود صندوق نصوص وزرين عليه للقيام بتجربة الفئة الجديدة حيث سنقوم بتعريف متغير خاص على مستوى النموذج باسم MyTimer من نوع فئتنا MyStopWtach وباستخدام العبارة WithEvents التي ستمكننا من استقبال الأحداث التي ستطلقها فئتنا

```
Private WithEvents MyTimer As New MyStopWtach
```

وسيكون كود الزرين من أجل بدء وإيقاف المؤقت باستخدام فئتنا السابقة كما يلي

```
Private Sub Button1_Click() Handles Button1.Click
    MyTimer.StartTimer()
End Sub

Private Sub Button2_Click() Handles Button2.Click
    MyTimer.StopTimer()
End Sub
```

الآن أنشئ معالج للحدث ReturnVlaue العائد للمتغير MyTimer واجعله بحيث يكون الكود فيه كالتالي ثم جرب تشغيل البرنامج فستحصل على رسالة مشابهة للرسالة في بداية المقال

```
Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
      ByVal e As MyStopWtach.ReturnValueEventArgs) Handles MyTimer.ReturnValue

    Me.TextBox1.Text = e.ReturnVlaue

End Sub
```

ولمعالجة هذه النقطة والتخلص من رسالة الخطأ سنحتاج لعمل Invoke للإجراء MyTimer_ReturnValue حتى نستطيع استخدام القيم المعادة منه في ضبط قيم خصائص التحكمات على النموذج وفي حالتنا هنا الخاصية Text لصندوق النصوص والعملية ببساطة ستتم كما يلي

في قسم المتغيرات العامة في النموذج سنقوم بتعريف إجراء مفوض Delegate يحمل نفس توقيع الإجراء MyTimer_ReturnValue وبدون جسم للإجراء كما يلي

```
Private Delegate Sub MyTimer_ReturnValueDelegate(ByVal sender As Object, _
                             ByVal e As MyStopWtach.ReturnValueEventArgs)
```

ويكون الكود الذي سينفذ المهمة بصورة صحيحة كما يلي

```
Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
        ByVal e As MyStopWtach.ReturnValueEventArgs) Handles MyTimer.ReturnValue

    If Me.TextBox1.InvokeRequired = True Then
        Dim d As New MyTimer_ReturnValueDelegate(AddressOf MyTimer_ReturnValue)
        Me.Invoke(d, New Object() {sender, e})
    Else
        Me.TextBox1.Text = e.ReturnVlaue
    End If
End Sub
```

حيث فحصنا قيمة الخاصية القابلة للقراءة فقط InvokeRequired لصندوق النصوص فإن كان False نقوم بضبط قيمة الخاصية Text باستخدام القيمة المعادة من الحدث مباشرة بدون أي مشاكل وإن كانت True عندها لن نستطيع ضبط القيمة مباشرة كي لا نحصل على الخطأ الوارد في بداية المقال عندها سنعرف متغير d من نوع الإجراء المفوض MyTimer_ReturnValueDelegate ونمرر له عنوان الإجراء MyTimer_ReturnValue ثم استخدمنا الطريقة Invoke العائدة للنموذج لتنفيذ نسخة آمنة من الحدث MyTimer.ReturnValue تمكننا من ضبط القيم المعادة إلى التحكمات وذلك بتمرير المتغير d كمحدد أول للخاصية Invoke ويكون المحدد الثاني للخاصية Invoke هو مصفوفة من النوع Object يتم تمرير محددات الإجراء MyTimer_ReturnValue كعناصر لها

وفيما يلي الكود الكامل للنموذج

```
Public Class Form1

    Private Delegate Sub MyTimer_ReturnValueDelegate(ByVal sender As Object, _
                             ByVal e As MyStopWtach.ReturnValueEventArgs)
```

```
    Private WithEvents MyTimer As New MyStopWtach

    Private Sub Button1_Click() Handles Button1.Click
        MyTimer.StartTimer()
    End Sub

    Private Sub Button2_Click() Handles Button2.Click
        MyTimer.StopTimer()
    End Sub

    Private Sub MyTimer_ReturnValue(ByVal sender As Object, _
             ByVal e As MyStopWtach.ReturnValueEventArgs) Handles _
              MyTimer.ReturnValue

        If Me.TextBox1.InvokeRequired = True Then
            Dim d As New MyTimer_ReturnValueDelegate( _
             AddressOf MyTimer_ReturnValue)

            Me.Invoke(d, New Object() {sender, e})
        Else
            Me.TextBox1.Text = e.ReturnVlaue
        End If
    End Sub

End Class
```

القسم العاشر - المجمعات والمشاريع

ويضم المواضيع التالية:

- مجمعات الدوت نيت Dot net Assemblies
- المجمعات ذات الأسماء القوية Strong Named Assemblies
- Friend Assemblies
- مترجم سطر الأوامر الخاص بفيجول بايزيك 2008
- التعامل مع محددات سطر الأوامر CommandlineArgs
- الإعدادات من وجهة نظر VB .net من 2002 حتى 2005
- ApplicationEvents
- كيفية استخدام ملف التعريف الخاص بالتطبيق لاستهداف نسخة معينة من الفريموورك
- كيف نقوم بتوزيع مشروعنا باستخدام تقنية Click Once
- نشر مشروعك باستخدام SetupWizard

مجمعات الدوت نيت Dot net Assemblies

ما هو المجمع

المجمع هو وحدة توزيع ويكون في أغلب الحالات عبارة عن ملف وهو يمثل الكود المترجم للتطبيق فأي كود سوف تكتبه سيتم تخزينه في النهاية في ملف EXE إن كان تطبيقا أو ملف DLL إن كان مكتبة أو توسعا ويحتوي المجمع على كل ما يحتاجه الدوت نيت لتحميل وتشغيل التطبيق.

والمجمعات تكون عامة Public أو خاصة Private. فالمجمعات الخاصة تكون مصممة للاستخدام ضمن تطبيق مفرد فقط وإن لم يكن هناك أية مكتبات DLL فالمجمع التنفيذي EXE يكون التطبيق وتظهر المجمعات الخاصة في المجلد الخاص بها – مجلد التنصيب الخاص بالتطبيق أو المكتبة – ويمكنك تشغيل مجمعين خاصين بنفس الوقت بدون أن يزعج أحدهما الآخر ويكون هذا صحيحا طالما أن كل مجمع يستخدم نفس التركيبة من مجالات الأسماء والفئات من أجل العناصر المكودة فيه فإن امتلك تطبيقان مختلفان فئة باسم WindowsApplication1.Class1 فلن يحدث تداخل بينهما عندما يتم تشغيل كلا التطبيقين لأنهما خاصين فالخاص يعني خاص.

وصممت المجمعات العامة Public ليتم مشاركتها عبر عدة تطبيقات دوت نيت وتختلف المجمعات العامة عن الخاصة بنقطتين:

- المجمعات العامة تمتلك دوما أسماء قوية strong Names إضافة إلى توقيع رقمي مشفر مرتبط بالمجمع وذلك لضمان أنه أتى من ذلك الموزع أو المصدر المسمى كما يمكن للمجمعات الخاصة أن تمتلك اسما قويا ولكن ذلك ليس ضروريا ويبنى الاسم القوى من أجل نسخة المجمع ورقم الإصدار ومعلومات الثقافة والتوقيع الرقمي وتتضمن الفريموورك أداة لتوليد الأسماء القوية sn.exe تساعدنا في هذه العملية ويتضمن فيجول ستوديو خيارا يمكنك من إضافة التوقيع الرقمي خلال عملية الترجمة
- وتخزن المجمعات العامة في المخزن العام للمجمعات GAC عادة ولكن ذلك لا يمنعك من وضع نسخة من مكوناتك المشتركة في مجلد التنصيب الخاص بتطبيقك ولن تصبح هذه المجمعات مشتركة بحق إلا عندما يتم وضعها داخل الـ GAC ويتواجد الـ GAC في مجلد مسمى assembly داخل مجلد الويندوز وطالما أنه تمت إضافة اسم قوي لمجمع الدوت نيت يمكنك إضافته إلى الـ GAC إما بنسخه للمجلد assembly أو باستخدام الأداة gacutil.exe

كما تسمح لك الدوت نيت بتنصيب عدة إصدارات من نفس المجمع في نفس النظام واستخدام جميع هذه الإصدارات بنفس الوقت من خلال عملية تدعى versioning وهذا ينطبق على التطبيقات EXE والمكتبات DLL والمجمعات الخاصة والمشتركة في الـ GAC ولمعاينة ذلك افتح مستكشف ويندوز وانتقل للمجلد assembly داخل مجلد الويندوز وبتصفح الملفات بعد فرزها حسب اسم المجمع – يظهر ذلك بشكل أوضح عند تنصيب الفريموورك 3.5 – فعند استعراض الملفات الموجودة هناك ستجد أن بعض الملفات مكررة أكثر من مرة كل مرة برقم إصدار مختلف.

ومع أنه في العادة هناك علاقة واحد إلى واحد بين الملفات والمجمعات إلا أنه هناك العديد من الحالات التي يكون فيها المجمع مكون من عدة ملفات وتقوم الدوت نيت بمراقبة هذه الملفات بشكل مستمر فإن طرأ أي تعديل عليها فسوف يتم إعلامك بذلك.

ماذا يوجد داخل المجمع

الملف EXE أو DLL للمجمع هو ملف تنفيذ محمول PE وهذه الملفات تمتلك نفس الصيغة للملفات ذات نفس النوع والغير معتمدة على الفريموورك وما يجعل ملفات PE مختلفة هي تلك الأشياء الإضافية التي تجدها بداخلها وككلمة عامة فالمجمع هو تجميع عدة أجزاء مختلفة في وحدة واحدة وتلك الأجزاء المختلفة في مجمعات الدوت نيت هي مصممة للدوت نيت خصيصا ويتكون ملفPE من ثلاثة أقسام رئيسية:

- الترويسة PE Header: وهي مطلوبة لجميع ملفات PE حيث يحدد هذا القسم موقع الأقسام الأخرى للملف
- قسم MSIL: وهو الكود الفعلي المرتبط بالمجمع مخزن بشكل نصف مترجم بحيث يصبح بلغة مايكروسوفت الوسيطة MSIL ولكن معالجات Intel و AMD في حاسوبك لا تستطيع معالجة كود MSIL مباشرة ولهذا أتت الفريموورك متضمنة مترجم تماما في الوقت JIT Compiler الذي يقوم بالترجمة الآنية للغة MSIL إلى كود يفهمه المعالج
- قسم Metadata كل تلك التفاصيل الإضافية التي يحتاجها الدوت نيت لمعرفة مجمعك تخزن في هذا القسم الأساسي حيث أن بعض هذه العناصر عند جمعها تشكل Assembly Manifest وهي نوع من الوثائق تصف المجمع بشكل عام للعالم حيث تحتوي الـ Metadata على العناصر التالية: - اسم المجمع – رقم الإصدار – محتويات الاسم القوي – إعدادات الثقافة واللغة – سرد أسماء ملفات المجمع – معلومات الأنواع المصدرة – المراجع المرتبطة – معلومات الأنواع الداخلية –

- والـ Masinfest هي الجزء الأهم من الـ Metadata وهي التعبير العام عن المجمع فعندما تنظر إلى الـ Mainfest ستعلم بلمحة على ماذا يحتوي المجمع وما هي متطلباته قبل أن تقوم بتحميله وتشغيله.

وحتى قبل وجود الدوت نيت كانت المكتبات والملفات التنفيذية تحتوي على بعض الـ Metadata ولكنها لم تكن تستخدم لتنسيق الوصول بين مختلف أقسام التطبيق ولا تقوم بتنظيم المعلومات الأساسية والموسعة وجاءت الـ Metadata في الدوت نيت لتضم جميع هذه العناصر ووجود كلا من الـ MSIL و الـ Metadata في كل مجمع يجعل هذه الملفات قابلة للقراءة والفهم باستخدام الأدوات المناسبة الأمر الذي ولد مشكلة فالشركات تستثمر الكثير من الوقت والمال في مجهود تطوير التطبيقات ولا ترغب بحصول أي نوع من الهندسة العكسية على ملفاتهم واستخلاص أكوادهم وأسرارهم. ولمنع القراءة العشوائية لمجمعات الدوت نيت قدمت مايكروسوفت وشركاءها الآخرين برمجيات obfuscator التي تقوم بخلط محتويات المجمع الأمر الذي يصعب على البشر القراءة والفهم ولكن ذلك لا يشكل أية صعوبة بالنسبة للدوت نيت فريموورك.

المجمعات ذات الأسماء القوية Strong Named Assemblies

يتضمن المجمع ذو الاسم القوي Mainfest تحدد الملفات التي يحتويها المجمع بالإضافة إلى توقيع رقمي يمكن لبرنامج آخر التحقق منه وذلك للتأكد من صحة التوقيع الرقمي الأمر الذي يؤكد أن المجمع لم يتم التلاعب به منذ إنشائه. ولكن ذلك لا يضمن بالتأكيد أن المجمع هو نفس ما تظنه فيمكن لمهاجم إنشاء مجمع جديد ثم القيام بتوقيعه رقميا وإن تأكدت من التوقيع يمكنك معرفة أن نسخة المهاجم لم يتم العبث بها منذ أن تمت كتابتها وأن الفيروسات المحتواة ضمن المجمع هي الفيروسات الأصلية.

ومع ذلك فتزويد مجمعاتك بأسماء قوية يضمن هويتها وهي تضمن بوسيلة ما أن هذا الكود يمكن الوثوق به ومن جهة أخرى تضمن الأسماء القوية هوية المجمعات بشكل فريد ولكن ذلك لا يضمن بالضرورة أنك قمت بكتابتها ومع ذلك يمكن تحديد قرار الوثوقية بشكل معقول بناء على هوية الاسم القوي لوحدها لأنه زودك بأن هذا المجمع معروف جيدا ورغم هذه المشكلة يعتبر التوقيع أفضل من لا شئ.

لتوقيع مجمع أنشئ مشروع فيجول بيزيك جديد كالعادة وفي مستكشف المشروع Project Explorer انقر نقرا مزدوجا على My Project لفتح خصائص المشروع ثم افتح صفحة Signing ثم قم بتحديد الخيار Sign the assembly ثم افتح القائمة المنسدلة الخاصة بـ Key File و اختر <New...> وفي صندوق الحوار الذي يظهر لك اختر الاسم الذي تريده للملف كما يمكنك إدخال كلمة سر بشكل اختياري ثم انقر OK. في هذه اللحظة يقوم فيجول ستوديو بإنشاء ملف Key يحتوي على جميع المعلومات اللازمة لتوقيع المجمع فعندما تقوم ببناء المجمع يستخدم فيجول ستوديو الملف Key لتوقيع المجمع الناتج.

وعندما يريد أي ملف تنفيذي استخدام المجمع – إن كان مكتبة Dll مثلا - يقوم بتحميله وينشئ جداول التهشير Hash اللازمة ويتأكد من أنها تطابق التوقيع وإن لم يكن مطابقا فالبرنامج يفترض أن المجمع قد تم العبث به أو أنه تالف ويرفض استخدامه.

ويضمن لنا توقيع المجمع أن لا أحد قد قام بالعبث به منذ أن تم إنشاؤه فإن قمت ببناء المجمع بنفسك فهذا يعطيك بعض الاطمئنان أن الكود آمن. ومع ذلك إن قام شخص آخر ببناء المجمع ووضعه على الويب فالتوقيع يضمن لك أن المجمع يحتوي على الأصل الذي وضعه الناشر ولكنه لا يضمن لك أن ذلك الناشر لم يقم بإدراج فايروس في ذلك المجمع ولا يضمن لك أيضا أن هاكرا ما لم يقم باستبدال المجمع الأصلي بنسخة أخرى من توقيعه. فإن قمت بكتابة مجمع تريد مشاركته مع الآخرين يمكنك جعل المجمع أكثر أمانا باستخدام سلطة الشهادات.

وسلطة الشهادات Certificate Authority هي شركة تبيع التواقيع الرقمية التي تضمن أصالة قطعة الكود الموقع وهذه الشركة تؤكد هويتك وبذلك يتأكد الآخرون بأنك فعلا من تدعي أن تكون فهي تعطيك شهادة Certificate يمكنك استخدامها لتوقيع المجمعات وعندما يستخدم أحد ما مجمعك لاحقا يمكن له التأكد من أن مجمعك هو المجمع الأصلي الذي قمت بإنشائه ولم يتم العيث به منذ أن قمت بكتابته إضافة إلى أنه يتأكد من صحة هويتك.

ولا يستخدم الهاكر سلطة الشهادات لنشر الفيروسات لأن ذلك يجعل من السلطة قادرة على التعرف عليهم مما يمكنك من تعقبهم وملاحقتهم قضائيا ومع ذلك فالشهادة لا تزال لا تضمن أن الكود ذات نفسه آمن ولكنها تشكل اعتمادية في حال تبين أن الكود يشكل خطورة

Friend Assemblies

الـ Friend Assembly هو مجمع Assembly مسموح له باستخدام العناصر الموجودة في مجمع آخر والتي تم تعريفها باستخدام محدد الوصول Friend فإن قمت بتحديد مجمع ما في فيجول بايزيك كـ Friend Assembly لم يعد متوجبا تعريف العناصر والأنواع في ذلك المجمع باستخدام محدد الوصول Public كي تتم رؤيتهم من المجمعات الأخرى وتعتبر هذه التقنية مريحة في حالتين:

- عندما تقوم بفحص كود يعمل في مجمع مختلف ولكنه يحتاج للوصول إلى عناصر في المجمع الذي يتم فحصه تم تعريفهم باستخدام محدد الوصول Friend
- عندما تقوم بتطوير مكتبة فئات Class Library وهناك إضافات لتلك المكتبة موجودة في مجمعات مختلفة ولكنها تتطلب الوصول لعناصر في المجمعات الموجودة تم تعريفهم باستخدام محدد الوصول Friend

يمكنك استخدام الصفة InternalsVisibleToAttribute لتحديد واحدة أو أكثر من الـ friend assemblies لمجمع محدد. فمثلا يمكنك تضمين الصفة InternalsVisibleToAttribute في المجمع A وتحديد المجمع B كـ friend assembly فسيكون بإمكان المجمع B الوصول لجميع الأنواع والعناصر في المجمع A التي تم تعريفها باستخدام محدد الوصول Friend كما يظهر في المثال التالي

```
Imports System.Runtime.CompilerServices

<Assembly: InternalsVisibleTo("FriendAssembliesB")>

' Friend class.
Friend Class FriendAssembliesA
    Public Sub Test()
        MsgBox("Friend Assemblies Sample Class")
    End Sub
End Class

' Public class with a Friend method.
Public Class FriendAssembliesClassA
    Friend Sub Test()
        MsgBox("Friend Assemblies Sample Method")
    End Sub
End Class
```

فجميع المجمعات التي يتم تعريفها بشكل واضح كـ Friend يمكنها الوصول إلى الأنواع والعناصر Friend فمثلا إن كان المجمع B هو Friend للمجمع A و المجمع C له مرجع إلى المجمع B فلن يكن لكلا المجمعين B و C حق الوصول للأنواع Friend في المجمع A

ويقوم المترجم بعمل تدقيقي أساسي لـ Friend Assemblies التي تم تمرير اسمها للصفة InternalsVisibleToAttribute فإن كان المجمع A يعرف المجمع B كـ Friend Assembly تكون قواعد التحقق كما يلي:

- إذا كان المجمع A يمتلك اسما قويا يجب على المجمع B أن يمتلك اسما قويا أيضا. فاسم الـ Friend Assembly الممرر إلى الصفة يجب أن يحتوي على اسم المجمع والمفتاح العام Public Key للاسم القوي Strong-Name Key الذي استخدم لتوقيع المجمع B. واسم الـ friend assembly الذي يتم تمريره للصفة InternalsVisibleToAttribute لا يمكن أن يكون الاسم القوي للمجمع B حيث أنك لا تضمن رقم النسخة للمجمع والثقافة والبناء architecture و ورمز المفتاح العام
- وإن لم يمتلك المجمع A اسما قويا يجب أن يحتوي اسم الـFriend Assembly على اسم المجمع فقط
- وإن كان للمجمع B اسما قويا يجب أن تحدد الـ strong-name key للمجمع B باستخدام خصائص المشروع أو عن طريق سطر الأوامر باستخدام خيار المترجم Keyfile/

مترجم سطر الأوامر الخاص بفيجوال بايزيك 2008

موجه سطر الأوامر الخاص بفيجوال ستوديو 2008

عندما نقوم بتنصيب .Net Framework 3.5 SDK أو إحدى نسخ Visual Studio 2008 يتم إنشاء مجموعة من المجلدات الجديدة التي تحتوي على مجموعة متنوعة من أدوات تطوير الدوت نيت والعديد من هذه الأدوات يتم تنفيذها من سطر الأوامر فإن كنت تريد تشغيل هذه الأدوات من أية نافذة سطر أوامر في الويندوز فيجب عليك عندها تسجيل هذه المجلدات ضمن متغير البيئة PATH العائد لنظام التشغيل ورغم أنه يمكنك تحديث ذلك المتغير بنفسك يدويا إلا أنه يمكنك توفير العناء على نفسك وتشغيل Visual Studio 2008 Command Prompt من المجلد Visual Studio Tools الكائن في المجلد Microsoft Visual Studio 2008 في قائمة ابدأ.

وتكمن الفائدة من فتح موجه سطر الأوامر بهذه الطريقة في كونه مضبوطا سلفا ليمكنك من الوصول إلى جميع أدوات التطوير العائدة للدوت نيت بدون أن تحتاج للقيام بتعديل متغيرات البيئة الخاصة بالنظام يدويا فإن أردنا رؤية المساعدة الخاصة بأداة ترجمة مشاريع VB2008 يمكنك تنفيذ الأمر التالي من موجه سطر الأوامر الخاص بفيجوال ستوديو 2008

```
Vbc -?
```

مترجم سطر الأوامر الخاص بفيجوال بايزيك 2008 vbc.exe

هناك العديد من التقنيات التي يمكنك استخدامها لترجمة الكود المصدري لفيجول بايزيك 2008 فبالإضافة لبيئة التطوير وبعض الأدوات من شركات أخرى تتعامل مع واجهة الدوت نيت يمكنك إنشاء مجمعات assemblies باستخدام مترجم سطر الأوامر الخاص بفيجوال بايزيك 2008 ومع أنه ربما لن تقوم ببناء البرامج على مستوى واسع باستخدام مترجم سطر الأوامر إلا أنه يبقى من الضروري أن تفهم كيف يمكنك ترجمة ملفاتك من النوع *.vb يدويا ويمكننا هنا ذكر بعض أسباب احتياجك للإلمام بهذه العملية:

- السبب الواضح الأول هو حقيقة أنه ربما لا يكون لديك نسخة من فيجول ستوديو 2008
- قد تكون في مؤسسة تعليمية تمنعك من توليد الكود باستخدام واجهات التطوير الرسومية
- قد تريد إنشاء أدوات ترجمة آلية للدوت نيت
- قد تريد تعميق فهمك لـ VB2008 فعندما تقوم باستخدام واجهات التطوير الرسومية فأنت تستخدم ضمنا vbc.exe وأنت بهذا تتعرف على ما يحدث وراء الكواليس عند عملية الترجمة
- فائدة إضافية وهي أنك تصبح مرتاحا مع استخدام أدوات الدوت نيت التي تعمل من موجه سطر الأوامر

بناء تطبيقات فيجول بايزيك 2008 باستخدام vbc.exe

لاستعراض كيفية بناء تطبيق دوت نيت بدون استخدام واجهة التطوير الرسومية سنقوم ببناء مجمع مؤلف من ملف واحد TestApp.exe وذلك باستخدام مترجم سطر الأوامر الخاص بفيجوال بايزيك 2008 مع الـ Notepad وسنحتاج في البداية هنا لبعض الكود المصدري ولهذا الغرض قم بفتح الـ notepad وأدخل الكود التالي

```
' A Simple Vb 2008 application
Imports System
Module TestApp
  Sub Main()
    Console.WriteLine("Testing! 1, 2, 3")
  End Sub
End Module
```

وبعد انتهائك احفظ الملف باسم TestApp.vb في مجلد ملائم مثلا C:\VbcExample

دعنا الآن نتعرف على الخيارات الأساسية لمترجم VB2008 حيث تكون نقطة الاهتمام الأولى هي تحديد اسم المجمع الذي سيتم إنشاؤه حيث يكون كل احتمال ممثل بقيمة ممررة إلى vbc.exe عبر محددات سطر الأوامر وفيما يلي أهم هذه الخيارات وشرحها

- /out اسم المجمع الذي سيتم إنشاؤه حيث تكون القيمة الافتراضية هي اسم الملف *.vb الأول
- /target:exe يقوم ببناء تطبيق كونسول تنفيذي وهو الخيار الافتراضي
- /target:library يقوم ببناء ملف *.dll وحيد
- /target:module يقوم ببناء module وهي وحدة تكوين المجمعات متعددة الملفات

- target:winexe/ يقوم ببناء تطبيق ويندوز تنفيذي

ورغم أنه يمكنك بناء تطبيقات ويندوز باستخدام target:exe إلا أن استخدام target:winexe يمنع نافذة الكونسول من الظهور عند تشغيل البرنامج. ولترجمة TestApp.vb إلى تطبيق كونسول باسم TestApp.exe افتح موجه سطر الأوامر الخاص بفيجوال ستوديو 2008 وانتقل إلى المجلد الذي قمت بحفظ الملف المذكور فيه

```
cd C:\VbcExample
```

ثم قم بإدخال الأمر التالي

```
vbc /target:exe TestApp.vb
```

وهنا لم نستخدم الخيار out لهذا سيتم تسمية الملف التنفيذي TestApp.exe تلقائيا ليوافق اسم ملف الكود المصدري المدخل فإن أردنا تسمية الملف التنفيذي باسم مختلف يمكننا إدخال الأمر التالي عند سطر الأوامر

```
vbc /target:exe /out:MyFirstApp.exe TestApp.vb
```

كما تجدر ملاحظة أن بعض خيارات مترجم VB2008 يكون لها نسخة مختصرة مثل t بدلا من target لذا يمكنك اختصار الإدخال على لوحة المفاتيح بالشكل التالي

```
vbc /t:exe TestApp.vb
```

وبما أن بعض هذه الخيارات افتراضية مثل t:exe لذا يمكن حذفها وبالتالي يمكن ترجمة الملف TestApp.vb كما يلي

```
vbc TestApp.vb
```

الوصول إلى مجمعات خارجية باستخدام vbc.exe

لاستعراض عملية الوصول إلى مجمعات خارجية سنقوم بتعديل الملف TestApp.vb ليقوم بعرض صندوق الرسائل الخاص بنماذج ويندوز ولعمل هذا افتح الملف TestApp.vb وعدله لتصبح محتوياته كالتالي

```
'A simple vb2008 appilication
Imports System

'Add This
Imports system.Windows.Forms

Module TestApp
  Sub Main()
    Console.WriteLine("Testing! 1, 2, 3")

    'Add This
    MessageBox.Show("Hello!")
  End Sub
End Module
```

لاحظ أنه لإضافة مرجع لمجال الأسماء System.Windows.Forms في VB2008 يتم استخدام الكلمة المحجوزة Imports وعند سطر الأوامر عليك إعلام vbc.exe عن أي مجمع يحتوي على مجالات أسماء مستوردة ولهذا كي تستطيع استخدام الفئة MessageBox يجب عليك استخدام الخيار reference لتحديد مرجع للمجمع System.Windows.Forms.DLL كما يلي

```
vbc /r:System.Windows.Forms.dll TestApp.vb
```

فإن قمت بتشغيل التطبيق الآن ستلاحظ ظهور صندوق الرسائل المحتوى في مجال الأسماء المضاف.

وإن كنت بحاجة للوصول إلى عدة مجمعات خارجية باستخدام vbc.exe عندها عليك ذكر أسماء تلك المجمعات بقائمة ضمن سطر الأوامر مفصولة بفاصلة كما في المثال

```
vbc /r:System.Windows.Forms.dll, System.Drawing.dll *.vb
```

ترجمة عدة ملفات مصدرية باستخدام vbc.exe

تم إنشاء النسخة الحالية من التطبيق TestApp.exe بواسطة ملف vb.* وحيد والحقيقة هي أن معظم المشاريع تكون مؤلفة من عدة ملفات vb.* وذلك يبقي الكود أكثر مرونة وتنظيما وسنفترض أنه لديك فئة إضافية موجودة في الملف HelloMsg.vb ومحتويات الملف كانت كما يلي

```
Imports System
Imports System.Windows.Forms

Class HelloMessage
   Sub Speak()
      MessageBox.Show("Hello Again")
   End sub
End class
```

وبافتراض أنك قمت بحفظ الملف HelloMsg.vb في نفس المجلد الذي قمت بحفظ الملف TestApp.vb فيه. قم بتعديل الكود في TestApp.vb ليستخدم الفئة الجديدة كما يلي

```
'A simple vb2008 appilication
Imports System

'Don't need This any More
'Imports system.Windows.Forms

Module TestApp
   Sub Main()
      Console.WriteLine("Testing! 1, 2, 3")

      'Don't need This any More either
      'MessageBox.Show("Hello!")

      'Exercise the HelloMessage Class!
      Dim hello as new HelloMessage()
      hello.Speak()
   End Sub
End Module
```

يمكنك الآن ترجمة ملفات vb.* كما يلي

```
vbc /r:System.Windows.Forms.dll TestApp.vb HelloMsg.vb
```

كما يمكنك مترجم VB2008 من استخدام المحارف البديلة * و ؟ في سطر الأوامر لإخبار vbc.exe بأنك تريد استخدام جميع ملفات vb.* المحتواة في مجلد المشروع كما يلي

```
vbc /r:System.Windows.Forms.dll *.vb
```

العمل مع ملفات الاستجابة الخاصة بـ vbc.exe

إذا كنت تريد بناء تطبيق VB2008 معقد يصبح عندها استخدام سطر الأوامر صعبا وخاصة عندما يكون عدد المجمعات وملفات vb.* كبيرا وقد تم تسهيل الأمر عليك هنا باستخدام ملفات الاستجابة response files وهي تحتوي على جميع التعليمات التي ستستخدم في عملية البناء الحالية وهذه الملفات تمتلك اللاحقة rsp ويمكننا إنشاء ملف استجابة لتطبيقنا السابق بالمحتويات التالية مع ملاحظة أن أسطر التعليقات تبدأ بالمحرف #

```
# This is the response file
#for the TestApp.exe app

#External assembly references.
/r:System.Windows.Forms.dll

#output and files to compile (using wildcards syntax).
/t:exe /out:TestApp.exe *.vb
```

وسنفترض أنه قد تم حفظه مع الملفات السابقة بنفس المجلد عندها يمكننا القيام بترجمة المشروع باستخدام ملف الاستجابة وذلك بكتابة @ يتبعها اسم ملف الاستجابة في سطر أوامر vbc.exe

```
vbc @TestApp.rsp
```

كما يمكنك تحديد عدة ملفات استجابة كدخل

```
vbc @FirstFile.rsp @SecondFile.rsp @ThirdFile.rsp
```

فعندما تستخدم هذه الطريقة يجب عليك ملاحظة أن المترجم يستخدم محددات سطر الأوامر بنفس ترتيب ورودها ولهذا فأي أمر في ملف rsp لاحق قد يتجاوز خيارات موجودة في ملفات rsp سابقة ولاحظ أيضا أن المحددات المكتوبة في سطر الأوامر قبل ملف الاستجابة سوف يتم تجاوزها باستخدام الأوامر الموجودة في ملف rsp ولهذا فإن قمت بإدخال

```
vbc /out:MyCoolApp.exe @TestApp.rsp
```

فسوف يبقى اسم المجمع TestApp.exe وذلك بسبب المحددات الموجودة في الملف TestApp.rsp وإن قمت بوضع المحددات بعد ملف rsp في سطر الأوامر فستقوم تلك المحددات بتجاوز الأوامر الموجودة في ملف rsp فإن قمت بإدخال السطر التالي فسوف يصبح اسم المجمع الناتج MyCoolApp.exe

```
vbc @TestApp.rsp /out:MyCoolApp.exe
```

ملف الاستجابة الافتراضي

آخر نقطة سيتم التحدث عنها هنا هو أن مترجم VB2008 مرتبط بملف استجابة افتراضي اسمه vbc.rsp وهو متواجد في نفس المجلد الموجود فيه vbc.exe حيث يمكنك فتحه والإطلاع على محتوياته باستخدام notepad فعندما تقوم ببناء تطبيق VB2008 باستخدام vbc.exe فسوف يتم استخدام ملف الاستجابة الافتراضي كمرجع حتى عندما تقوم باستخدام ملفات استجابة مخصصة وإن كنت لسبب ما لا ترغب باستخدام ملف الاستجابة الافتراضي يمكنك استخدام الخيار noconfig كما يلي

```
vbc @TestApp.rsp /noconfig
```

خاتمة

كما يمتلك مترجم سطر أوامر VB2008 العديد من الخيارات الأخرى التي يمكن استخدامها للتحكم بمجمع دوت نيت المولد وعند هذه النقطة يجب أن تكون قد أصبحت ملما بالأساسيات التي تمكنك من استخدام باقي الخيارات التي يمكنك الإطلاع عليها من خلال الوثائق المرفقة مع بيئة التطوير

التعامل مع محددات سطر الأوامر CommandlineArgs

الخاصية CommandLineArgs

يمكننا التعامل مع محددات سطر الأوامر من خلال الخاصية CommandLineArgs التي تعيد مجموعة نصية للقراءة فقط ReadOnly Collection Of String تحتوي على محددات سطر الأوامر وتكون صيغتها العامة على الشكل

```
Public ReadOnly Property CommandLineArgs() As ReadOnlyCollection(Of String)
```

وهي متواجدة في المجمع Microsoft.VisualBasic.dll وضمن مجال الأسماء Microsoft.VisualBasic.ApplicationServices
وهي بالنسبة لتطبيقات الكونسول ConsoleApplicationBase.CommandLineArgs وبالنسبة لتطبيقات الويندوز My.Application.CommandLineArgs

وتجدر ملاحظة أنه عند تشغيل عدة نسخ من التطبيق في آن واحد فإن هذه الخاصية تعيد القيم الخاصة بأول مرة تم تشغيل البرنامج فيها وللحصول على محددات سطر الأوامر للمرات التالية التي تم تشغيل البرنامج فيها علينا معالجة الحدث My.Application.StartupNextInstance وفحص الخاصية CommandLine العائدة لـ StartupEventArgs – وهنا أنصحك بمراجعة موضوعي أحداث التطبيق – وتجدر ملاحظة الاختلاف بين الخاصية CommandLineArgs التي تعيد محددات سطر الأوامر فقط والخاصية CommandLine التي تعيد سطر الأوامر التي تفذ من خلاله التطبيق كاملا بما فيه اسم الملف التنفيذي للتطبيق وهذه الملاحظة خاصة بالبرنامج وحيد التواجد Single Instance Application أي البرنامج المفترض تشغيل نسخة واحدة فقط منه وبالمناسبة نحن نقوم بعمل برنامج وحيد التواجد بوضع إشارة في المربع بجانب Make Single Instance Appliocation في صفحة Application في خصائص My Project

وهذه الخاصية ليست متوفرة لجميع أنواع التطبيقات التي يمكنك إنشاؤها فهي متوفرة لأنواع التطبيقات التالية فقط Windows Application و Console Application و Windows Service فقط وقد تحتاج في بعض الحالات للفئة EnvironmentPermission من أجل الحصول على الصلاحيات الكافية لقراءة محددات سطر الأوامر من خلال هذه الخاصية وربما سأقوم مستقبلا بعمل موضوع مستقل عن الفئة EnvironmentPermission

أمثلة عن الاستخدام

إذا أردنا معالجة محدد ما المفترض أن يمرر عبر سطر الأوامر /input= مثلا يمكننا استخدام الكود التالي لإظهار بقية القيمة الممررة

```
Private Sub ParseCommandLineArgs()
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

أو يمكننا عمل إجراء لمعالجة عدة محددات لسطر الأوامر

```
For Each cr In My.Application.CommandLineArgs
    HandleArgs(cr.ToUpper)
Next

Private Sub HandleArgs(ByVal carg As String)
    Select Case carg
        Case "/S"
            ' معالجة المحدد /S

        Case "/T"
            ' معالجة المحدد /T
        Case Else
            ' معالجة أي محدد آخر

    End Select
End Sub
```

وفي حالة تشغيل عدة نسخ من التطبيق في آن واحد هذا مثال عن معالجة الحدث StartupNextInstance حيث يمكننا الوصول للحدث من خلال فتح خصائص My Project ومن صفحة Application اضغط الزر View Application Events لينقلك لمحرر الكود المناسب

```
Private Sub MyApplication_StartupNextInstance( _
    ByVal sender As Object, ByVal e As _
    Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs _
    ) Handles Me.StartupNextInstance

    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In e.CommandLine
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

الإعدادات من وجهة نظر VB .net من 2002 حتى 2005

العديد من التطبيقات تحتاج لتخزين إعدادات محددة لمتابعتها بين الجلسات. ولكن كيف ستقوم بتخزين هذه الإعدادات ضمن التطبيقات المكتوبة ضمن الفريموورك.؟ ليس من السهل دوما الإجابة الصحيحة عن هذا السؤال وستجد مجالا متنوعا من الحلول لهذه المسألة من العديد من المصادر ولكن قليلا منها يمثل الطريقة المثلى لمعالجة هذه الحاجة

دعنا أولا نعرف نوعين رئيسيين لهذه الإعدادات: إعدادات خاصة بالتطبيق وإعدادات خاصة بالمستخدم فإعدادات التطبيق يتم توزيعها مع البرنامج وهي تؤثر على جميع المستخدمين ويكون المسؤول عن تغييرها في العادة مدراء الأنظمة عندما نريد تغيير التصرف العام للتطبيق. فمثلا برنامج لتنظيم الأيدي العاملة عندما يتم تطبيقه في مؤسسة ما سيتم ضبط إعداداته العامة مثل إعداد مخدم قاعدة البيانات التي سيتصل بها المستخدمون وإضافة شعار الشركة لواجهة البرنامج وهكذا. إعدادات التطبيق هذه لا علاقة لها بالمستخدم ويمكن أن يتم إعدادها في نفس المجلد الذي سيتم تثبيت التطبيق فيه وبشكل يكون فقط مدير النظام له صلاحيات التعديل والحذف على هذه الإعدادات وفي الحالات العامة سيحتاج التطبيق لقراءة هذه الإعدادات دون الحاجة لتعديلها

وإعدادات المستخدم تكون خاصة بكل مستخدم على حد ى ويجب أن تكون قابلة للتحرير من داخل الكود فمن أجل تطبيق ما يمكن أن تضم إعدادات المستخدم حجم ومكان نافذة التطبيق وحالة البدء للتطبيق وتغيير هذه المعلومات بسرعة. كل هذه المعلومات يجب استعادتها عند بدء جلسة التطبيق وتحريرها عند الضرورة أثناء تشغيل التطبيق وحفظها للقرص لتكون متوفرة في المرة القادمة التي يقوم المستخدم بتشغيل التطبيق.

ضبط واستعادة إعدادات المستخدم

أول خطوة هي إنشاء ملف إعداد داخل تطبيقك باختيار Add New Item من قائمة Project و اختيار Application Configuration File و اترك الاسم الافتراضي للملف App.Config كما هو دون تغيير حيث سيتم نقله وإعادة تسميته تلقائيا بالشكل المناسب عند قيامك ببناء التطبيق ففي الملف الجديد يمكنك تحديد إعدادات مخصصة أو إضافة إعداداتك الخاصة ضمن القسم appSettings حيث تكون عبارة عن أزواج اسم/قيمة وتخزين بعض المعلومات البسيطة قد ينتج ملفا شبيها بالتالي

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="Company Name" value="Java Jitters" />
        <add key="Database Server" value="BigSQLServer" />
    </appSettings>
</configuration>
```

ومن السهل استعادة هذه القيم عند الحاجة من خلال الفئات والإجراءات الموجودة ضمن النطاق System.Configuration وحتى أنك لن تحتاج لتحميل الملف بنفسك حيث سيتم اكتشافه وتحميله تلقائيا إن كان يحمل الاسم المناسب وموضوعا في المكان المناسب

```
Dim companyName As String _
    = ConfigurationSettings.AppSettings("Company Name")
```

لماذا لاتستخدم App.Config ؟

كثيرا ما نرى السؤال كيف أعمل ملف App.Config الخاص بي بعد أن استخدموا ملف التعريف ورأوا عملية البرمجة السهلة المتعلقة به لماذا لا يمكنهم استخدام نفس الملف لتخزين إعدادات المستخدم الخاصة بهم لماذا يوجد طريقتين مختلفتين للإعدادات وذلك بعد اكتشافهم أن المجال System.Configuration لا يقدم طريقة لكتابة الملف app.config والتي تعتبر بمثابة تحذير ولكنها لن تقف عائقا أمام المبرمج المصمم وبعد البحث في العادة يقومون باستخدام ملف عادي أو XmlDocument وهذه تعتبر فكرة غير جيدة ولا يجب على تطبيقك كتابة هذه الملفات لعدة أسباب منها الصلاحيات، عدم عزل المستخدمين وكون الملف app.config جزءا من ملفات إعداد التطبيق لهذه الأسباب وأسبابا أخرى لا يجب عليك استخدام الملف app.config لغرض تخزين إعدادات المستخدم ولكن لا تقلق فتخزين إعدادات المستخدم ليس بتلك الصعوبة

إنشاء فئة إعدادات وتخزينها على القرص

المفتاح لإنشاء إعدادات المستخدم الخاصة بك هو إنشاء فئة Class تمثل جميع البيانات حيث سيكون واجهتك لإعدادات المستخدم الخاصة بك وهي أول شئ يجب عليك بناؤه عندما تريد تخزين أية معلومات كالفئة التالية مثلا

```
Imports System.Drawing
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Environment
Imports System.Collections.Specialized

Public Class Settings

    Private m_BackgroundColor As String
    Private m_RecentFiles As New StringCollection
    Private m_LastWindowBounds As Rectangle _
        = New Rectangle(0, 0, 200, 200)
    Private m_LastWindowState As Windows.Forms.FormWindowState _
        = FormWindowState.Normal

    <Xml.Serialization.XmlIgnore()> _
    Public Property LastWindowBounds() As Rectangle
        Get
            Return m_LastWindowBounds
        End Get
        Set(ByVal Value As Rectangle)
            m_LastWindowBounds = Value
        End Set
    End Property

    Public Property LastWindowPos() As Point
        Get
            Return m_LastWindowBounds.Location
        End Get
        Set(ByVal Value As Point)
            m_LastWindowBounds.Location = Value
        End Set
    End Property

    Public Property LastWindowSize() As Size
        Get
            Return m_LastWindowBounds.Size
        End Get
        Set(ByVal Value As Size)
            m_LastWindowBounds.Size = Value
        End Set
    End Property

    Public Property LastWindowState() As Windows.Forms.FormWindowState
        Get
            Return m_LastWindowState
        End Get
        Set(ByVal Value As Windows.Forms.FormWindowState)
            m_LastWindowState = Value
        End Set
```

```
    End Property

    Public Property RecentFiles() As StringCollection
        Get
            If m_RecentFiles Is Nothing Then
                m_RecentFiles = New StringCollection
            End If
            Return m_RecentFiles
        End Get
        Set(ByVal Value As StringCollection)
            If Value Is Nothing Then
                m_RecentFiles = New StringCollection
            Else
                M_RecentFiles = Value
            End If
        End Set
    End Property

    Public Property BackgroundColor() As String
        Get
            Dim colorToReturn As String

            If m_BackgroundColor Is Nothing OrElse _
                m_BackgroundColor = String.Empty Then
                colorToReturn = ColorTranslator.ToHtml( _
                    Color.FromKnownColor(KnownColor.Control))
            Else
                colorToReturn = m_BackgroundColor
            End If
            Return colorToReturn
        End Get
        Set(ByVal Value As String)
            If Not ColorTranslator.FromHtml(Value).IsEmpty Then
                m_BackgroundColor = Value
            Else
                m_BackgroundColor = String.Empty
            End If

        End Set
    End Property

End Class
```

بعد انتهاؤك من كتابة الفئة المناسبة لإعداداتك المطلوبة ستكون الخطوة التالية هي كتابتها وقراءتها من ملف على القرص وإحدى الطرق المناسبة هي الفئتين System.Xml.XmlReader و System.Xml.XmlWriter ولكنه يكون من الأسهل استخدام تسلسل XML Serialization حيث يكفل جميع الضروريات لإنشاء تواجد للفئة في ملف XML و استعادتها منه وهو قوي بحيث يستطيع التعامل مع المعلومات المنقوصة وحتى الخصائص الإضافية التي يمكن أن يتم إضافتها مستقبلا وبهذا يتضح أنه خيار جيد ويمكنك تخزين تواجد للفئة على القرص باستخدام الفئة System.Xml.Serialization.XmlSerializer كما يمكنك استخدام نفس الفئة للاستعادة لاحقا انظر المثال

```
Public Sub SaveSettings()
    Dim xs As New XmlSerializer(GetType(Settings))

    Dim sw As New IO.StreamWriter("C:\settings.xml")
    xs.Serialize(sw, currentSettings)
    sw.Close()
```

```
    Dim sr As New IO.StreamReader("C:\settings.xml")
    currentSettings = CType(xs.Deserialize(sr), Settings)
    sr.Close()

End Sub
```

في هذا المثال البسيط نريد وضع كود فعلي في فئة الإعدادات الخاصة بنا كدمج بين العديد من الإجراءات والطرق المختلفة وقد زودت بعض الخيارات المختلفة ليتم تقديمها ضمن التدفق stream حيث سيهتم بحفظها على القرص وعملية التسلسل/إلغاء التسلسل serializing or deserializing تتطلب أولا إنشاء تواجد للفئة XmlSerializer

```
Dim xs As New XmlSerializer(GetType(Settings))
```

وستحتاج أيضا إلى تواجد لـ TextWriter, or XmlWriter أو TextReader, or XmlReader لتمثل عملية الحفظ والتحميل

```
Dim sw As New IO.StreamWriter("C:\settings.xml")
```

ومكان تخزين الملف على القرص يعتبر من النقاشات الهامة وأقترح مجلد بيانات التطبيق للمستخدم أو أي مكان معزول آخر ويعتبر الخيار الأول هو الأفضل حيث يمكنك الحصول على مساره الكامل باستخدام Application.UserAppDataPath أو System.Environment.GetFolderPath حيث يمكن الحصول على المسار المطلوب بالعديد من الطرق ولكني أعتقد أن هذه الطريقة تعمل بشكل جيد كفاية

```
Dim filePath As String
filePath = Path.Combine(GetFolderPath( _
        SpecialFolder.ApplicationData), Application.CompanyName)
filePath = Path.Combine(filePath, Application.ProductName)
filePath = Path.Combine(filePath, FILENAME)
Dim settingsFile As New FileStream(filePath, FileMode.Create)
```

التخزين المنفصل

و المكان المثالي الآخر هو التخزين المنفصل حيث يكون له مكان خاص ضمن مجلد بيانات التطبيق للمستخدم الذي يوفر ملفات فريدا مربوطا بالمستخدم وببعض المعلومات حول التطبيق حيث يمكنك استخدام الفئات الموجودة ضمن System.IO.IsolatedStorage لإنشاء ملف أو فتح آخر موجود في تلك المنطقة

```
Dim ifs As IsolatedStorageFile
ifs = IsolatedStorageFile.GetUserStoreForAssembly()
Dim settingsFile As New IsolatedStorageFileStream( _
    FILENAME, FileMode.Create, ifs)
Return settingsFile
```

الحفظ والتحميل من داخل التطبيق

في النموذج الرئيسي لتطبيقك ستحتاج لتحميل ملف الإعدادات من القرص عند بدء تطبيقك وحفظها عند نهايته وكل الإجرائيات التي ستحتاجها موجودة في فئتك وسيتطلب ذلك العمل مع تلك الإجرائيات في الفئة لتحرير الإعدادات التي تريدها ففي تطبيق بسيط بنافذة واحدة سيكون من السهل انجاز هذا ببضعة سطور من الكود ومن أجل تحميل الإعدادات عند بدء التطبيق يمكنك النظر إلى المثال

```
Dim currentSettings As Settings

Public Sub LoadSettings()
    Try
        Me.currentSettings = Settings.Load()
        With Me.currentSettings
            Me.Bounds = .LastWindowBounds
            Me.BackColor = ColorTranslator.FromHtml(.BackgroundColor)
```

```
            Me.WindowState = .LastWindowState
        End With
        UpdateRecentFileMenu()
    Catch ex As Exception
        Me.currentSettings = New Settings
    End Try
End Sub

Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call
    LoadSettings()

End Sub
```

ولتحميلها يمكنك النظر للمثال

```
Protected Overrides Sub OnClosing(ByVal e As _
    System.ComponentModel.CancelEventArgs)
    SaveSettings()
End Sub

Private Sub SaveSettings()
    With Me.currentSettings
        .LastWindowBounds = Me.Bounds
        .BackgroundColor = ColorTranslator.ToHtml(Me.BackColor)
        .LastWindowState = Me.WindowState
        .PersistMe()
    End With
End Sub
```

ماذا عن فيجول بايزيك 2005

ناقشنا في هذا الموضوع كيف وأين تقوم بتخزين إعداداتك ولكن في VB2005 كثير من هذه المسائل تمت معالجتها من أجلك حيث تمت إضافة محرر للإعدادات في خصائص التطبيق يمكنك استخدامها بسهولة لضبط كلا من إعدادات المستخدم و إعدادات التطبيق بكل سهولة وبعد قيامك بضبطها بالشكل المناسب يمكنك قراءتها بكل سهولة

```
Me.Text = My.Settings.CompanyName
If My.Settings.StartMaximized Then
    Me.WindowState = FormWindowState.Maximized
End If
```

في الفريموورك 2 ستلاحظ سهولة إنشاء الإعدادات حيث لم يعد من الضروري فتح ملف الإعداد وإدخال إعداداتك الخاصة بدلا عن ذلك يمكنك التوجه إلى مصمم الإعدادات مباشرة وإضافة إعداداتك مهما يكن مجالها تطبيق أم مستخدم. حيث تكون الإعدادات على مستوى التطبيق للقراءة فقط وهي مشتركة بين جميع المستخدمين في التطبيق وتكون مخزنة في الملف app.config في القسم <applicationSettings> والذي يختلف عن القسم <appSettings> الموجود في الفريموورك 1 وفي وقت التشغيل سيكون الملف app.config في المجلد Bin وسوف يتم إعادة تسميته وفقا لاسم البرنامج وكمثال على إعدادات على مستوى التطبيق تعريف وصلة قاعدة البيانات أو عنوان خدمة ويب أو عنوان ملقم ... الخ

```
<applicationSettings>
```

```
    <MySettingsDemo.MySettings>
      <setting name="SMTPServerIP" serializeAs="String">
        <value>127.0.0.1</value>
      </setting>
      <setting name="MyServicesURL" serializeAs="String">
        <value>http://localhost/myservices.asmx</value>
      </setting>
    </MySettingsDemo.MySettings>
  </applicationSettings>
```

الإعدادات على مستوى المستخدم تكون مخصصة لكل مستخدم ويمكن قراءتها وإعادة ضبطها بأمان من خلال كود التطبيق وتكون مخزنة في ملف user.config ولكي نكون دقيقين هناك ملفان user.config لكل مستخدم لكل تطبيق واحد للمشاركة وواحد بدون مشاركة ومع ذلك تنص وثائق VB2005 على أن الملف user.config سيتم تسميته تبعا لاسم المستخدم ولكن هذا ليس دقيقا حيث سيتم تخزينه في المسار

```
<c:\Documents and Settings>\<username>\[Local Settings\]Application


Data\<companyname>\<appdomainname>_<eid>_<hash>\<verison>
```

حيث

```
<c:\Documents and Settings> مجلد بيانات المستخدم
<username> اسم المستخدم
<companyname> اسم الشركة
<appdomainname> Application Domain
<eid> معرف المستخدم
<hash> hash
<version> رقم نسخة التطبيق
```

وكمثال

```
C:\Documents and Settings\Emad.BROKENOAK\Local Settings\Application
Data\MySettingsDemo\MySettingsDemo_9cfe5ef1\1.0.0.0
```

حيث يتم إنشاء الملف user.config في أول مرة يتم فيها تشغيل التطبيق بواسطة مستخدم جديد والقيم الغير افتراضية سوف يتم وضعها في ملف على مستوى المستخدم بينما القيم الموجودة في app.config في القسم <userSettings> هي القيم الافتراضية لهذه الإعدادات وهي محددة أيضا في الملف app.config وفيما يلي كيف يمكن أن يبدو الملف app.config

```
  <userSettings>
    <MySettingsDemo.MySettings>
      <setting name="LastSearchedItem" serializeAs="String">
        <value />
      </setting>
      <setting name="FormSize" serializeAs="String">
        <value>400, 400</value>
      </setting>
      <setting name="FormLocation" serializeAs="String">
        <value>0, 0</value>
      </setting>
    </MySettingsDemo.MySettings>
  </userSettings>
```

الإعدادات على مستوى المستخدم عظيمة لتخزين إعدادات التطبيق حيث عادة ما تختلف من مستخدم لآخر وكمثال على ذلك إعدادات العرض كحجم الخط وموقع النافذة أو لائحة الملفات المفتوحة أخيرا وهكذا. وهذه البنية مرنة جدا لأنها تمكن تطبيقك من تخزين الإعدادات لتطبيقك حتى لو كنت تعمل في بيئة محدودة الصلاحيات.

إنشاء إعداد

لنفترض أنه لدينا صندوق بحث في تطبيقك وأنت تريد تخزين آخر قيم تم البحث عنها

١. في مستكشف الحل انقر نقرا مزدوجا على المشروع لعرض خصائص المشروع
٢. انقر صفحة Settings لعرض مصمم الإعدادات
٣. أدخل الاسم Name والنوع Type والمجال Scope للإعداد المطلوب
٤. في حقل القيمة Value أدخل القيمة الافتراضية لذلك الإعداد

وراء المشهد

عندما تضيف إعدادا جديدا في مصمم الإعدادات تجري عدة أشياء في الخلفية أولا يتم إضافة مدخل جديد في الملف app.config لتحديد هذا الإعداد

```
 <configSections>
    <sectionGroup
name="applicationSettings"type="System.Configuration.ApplicationSettingsGroup,
System,
Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
      <section name="MySettingsDemo.MySettings"
type="System.Configuration.ClientSettingsSection, System,
Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    </sectionGroup>
  </configSections>
```

ثانيا تعريف الإعداد ذات نفسه يتم إضافته للملف app.config بالاعتماد على نوع Type الإعداد الذي تحدده حيث يمكنك أن ترى قسم applicationSettings أو userSettings أو connectionStrings أو أكثر من واحد منها اعتمادا على الإعدادات التي تحددها كمثال:

```
<userSettings>
    <MySettingsDemo.MySettings>
      <setting name="LastSearchedItem" serializeAs="String">
        <value />
      </setting>
      <setting name="FormSize" serializeAs="String">
        <value>400, 400</value>
      </setting>
      <setting name="FormLocation" serializeAs="String">
        <value>0, 0</value>
      </setting>
    </MySettingsDemo.MySettings>
  </userSettings>
  <connectionStrings>
    <add name="MySettingsDemo.MySettings.ConnectionString"
connectionString="Server=TABLET; User ID=sa; Password=1234;
Database=Northwind; Persist Security Info=True"
providerName="System.Data.SqlClient" />
  </connectionStrings>
  <applicationSettings>
    <MySettingsDemo.MySettings>
      <setting name="SMTPServerIP" serializeAs="String">
        <value>192.168.2.11</value>
      </setting>
      <setting name="MyServicesURL" serializeAs="String">
```

```
      <value>http://localhost/MyServices/Service1.asmx</value>
    </setting>
  </MySettingsDemo.MySettings>
</applicationSettings>
```

الوصول للإعدادات

بعد إضافة الإعداد سنحتاج لقراءته وتحديثه حيث يمكنك الوصول إليه عبر My.Settings كمثال

```
My.Settings.LastSearchedItem
```

وكطريقة أخرى للوصول لهذه الإعدادات هي ربطها بإحدى خصائص التحكم حيث من صندوق الخصائص انتقل إلى قسم Data وفيه (ApplicationSettings) ومنها (PropertyBinding) ومن الصندوق الذي يظهر لك انتقل للخاصية التي تريد ربطها مثلا Location وقم باختيار الضبط المناسب لها مثلا FormLocation أو حتى يمكنك إنشاء إعداد جديد من هناك بالنقر على New وكنتيجة لهذا الربط سيقوم المصمم بإنشاء كود في الخلف كالتالي

```
Me.DataBindings.Add(New System.Windows.Forms.Binding("Location",
MySettingsDemo.MySettings.Value, "FormLocation", True,
MySettingsDemo.MySettings.Value.LastSearchedItem, Nothing,
System.Windows.Forms.BindingUpdateMode.OnPropertyChanged))
```

عندما تربط الإعدادات بهذه الطريقة لن تضطر للقلق حول قراءة وكتابة هذه الإعدادات حيث سيتم ضبط الخاصية عند الحاجة وسوف يتم تخزين قيمتها تلقائيا إذا تم تغييرها ويمكنك ملاحظة أن هذا كله يتم دون اضطرارك لكتابة أي سطر من الكود كما يجدر ملاحظة أن هذه الإعدادات يتم تخزينها تلقائيا في بعض أنواع المشاريع مثل Visual Basic Windows Forms ويجب أن تستدعي الإجراء Save من My.Settings في بعضها الآخر مثل مشاريع Class Library وإذا كانت الخاصية غير مدرجة ضمن نافذة الربط يمكنك تحقيق هذا الربط باستخدام الكود حيث لبعض الأسباب الخاصية Size غير متوفرة كخاصية يمكن ربطها في النسخة التجريبية وهي متوفرة في النسخة النهائية ويمكن تحقيق الربط بإضافة Binding جديد لمجموعة DataBindings collection ضمن النموذج حيث تستخدم الفئة Binding لتحديد اسم الخاصية واسم الإعداد وطريقة التحديث كمثال يمكننا ربط الخاصية Size بالإعداد FormSize يدويا كالمثال

```
Me.DataBindings.Add(New System.Windows.Forms.Binding("Size", _
                    MySettingsDemo.MySettings.Value, _
                    "FormSize"))
```

الإعدادات المتقدمة

إذا أردت تحكما إضافيا على إعداداتك يمكنك تحديد الفئة Class الخاصة بك والتي يجب أن تشتقها من الفئة ApplicationSettingsBase التي تشكل الفئة الأب لجميع فئات الإعدادات حيث تحدد كل إعداد كخاصية لهذه الفئة و تسبقها بالخاصية ApplicationScopedSettingAttribute إذا كانت على مستوى التطبيق أو بالخاصية UserScopedSettingAttribute إذا كانت على مستوى المستخدم وهذا ما يفعله مصمم الإعدادات بالضبط وهذا مثال من كود مولد بواسطة المصمم

```
Partial NotInheritable Class MySettings
    Inherits System.Configuration.ApplicationSettingsBase

    <System.Diagnostics.DebuggerNonUserCode(),  _
     System.Configuration.UserScopedSettingAttribute(),  _
     System.Configuration.DefaultSettingValueAttribute("400, 400")>  _
    Public Property FormSize() As System.Drawing.Size
        Get
            Return CType(Me("FormSize"),System.Drawing.Size)
        End Get
        Set
            Me("FormSize") = value
        End Set
    End Property

    <System.Diagnostics.DebuggerNonUserCode(),  _
```

```
System.Configuration.SpecialSetting(System.Configuration.SpecialSetting.Connecti
onString),  _
     System.Configuration.ApplicationScopedSettingAttribute(),  _
     System.Configuration.DefaultSettingValueAttribute("Server=TABLET;
User ID=sa; Password=1234; Database=Northwind; Persist Security In"& _
        "fo=True")>  _
    Public ReadOnly Property ConnectionString() As String
        Get
            Return CType(Me("ConnectionString"),String)
        End Get
    End Property

    <System.Diagnostics.DebuggerNonUserCode(),  _
     System.Configuration.ApplicationScopedSettingAttribute(),  _
     System.Configuration.DefaultSettingValueAttribute("192.168.2.11")>  _
    Public ReadOnly Property SMTPServerIP() As String
        Get
            Return CType(Me("SMTPServerIP"),String)
        End Get
    End Property
...
```

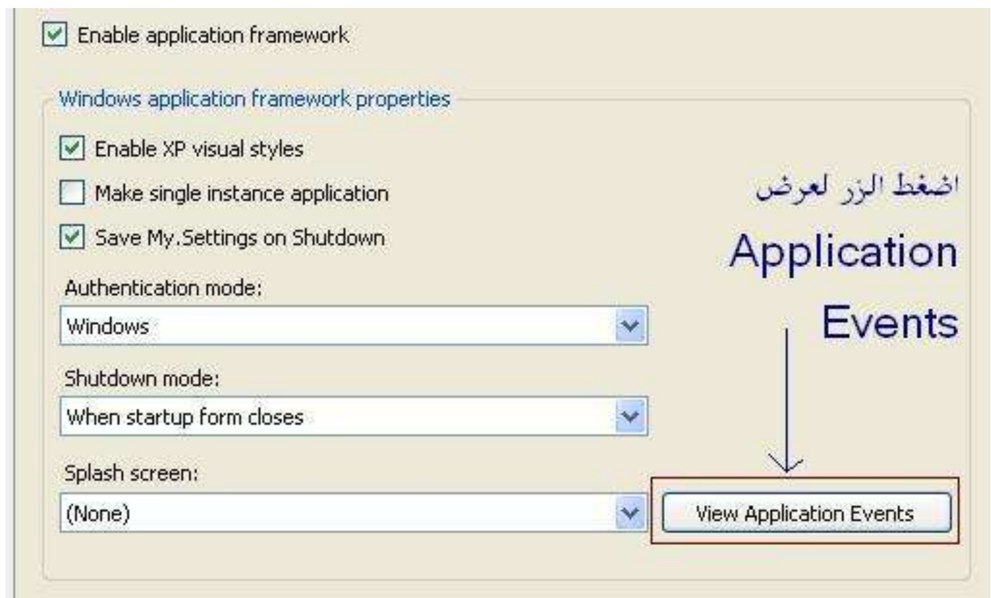
ويمكنك ملاحظة أن الـ ConnectionString محاط بالخاصية SpecialSetting التي يكون لها قسم خاص في ملف الإعدادات كما تجدر ملاحظة Partial class في بداية التعريف مما يعني أنه يمكنك إنشاء ملف جديد في مشروعك وتضمينه جزء آخر من أحد الفئات Class في مشروعك وبهذه التقنية يمكنك تعديل إعداداتك دون تعديل الكود المنشأ بواسطة المصمم ويمكنك إضافة معالجة للأحداث في هذه الفئة والتي تمنحك دقة أكبر في التحكم بهذه الإعدادات وهذه الأحداث هي SettingChanging و SettingsSaving و PropertyChanging

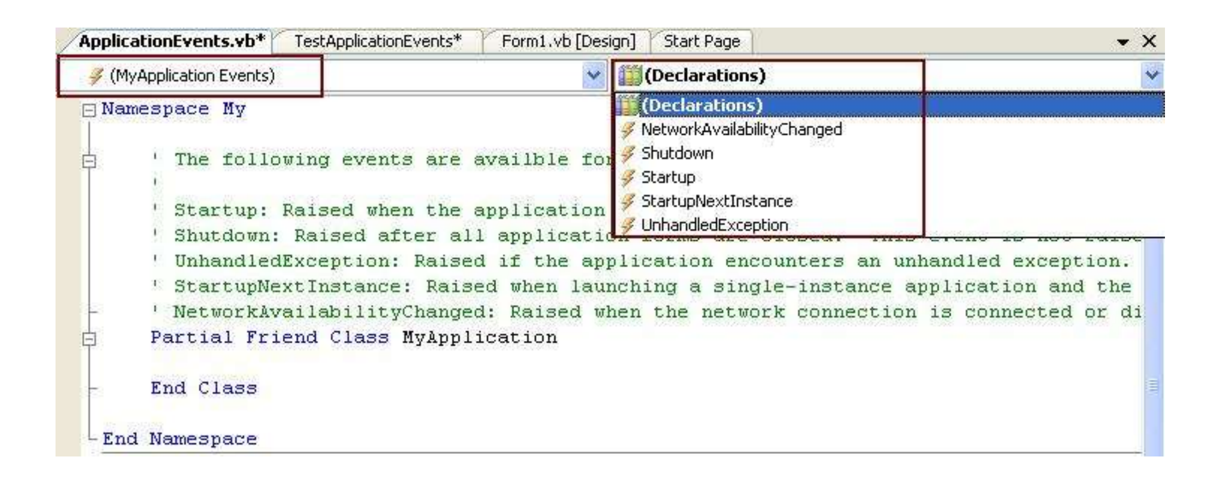
أحداث التطبيق Application Events

الشرح هنا خاص بـ Visual Basic 2005

للوصول إليها:

من مستكشف الحل Solution Explorer انقر بالماوس اليميني على مشروعك ثم اختر Properties ثم من صفحة Application اضغط زر View Application Events فيتم نقلك لمحرر الكود حيث تتمكن من استخدام الحدث الذي تحتاج لمعالجته في تطبيقك من خصائص التطبيق كما هو ظاهر بالصورة





وفيما يلي تعريف بكل حدث على حدة

My.Application.Startup Event

حيث ينطلق هذا الحدث عند بداية التطبيق لتضع فيه الأوامر التي تريد تنفيذها عند بداية التطبيق فمثلا إذا أردت تغيير الثقافة الخاصة بالتطبيق إلى العربية – سورية نستخدم كودا شبيها بالتالي:

```
Private Sub MyApplication_Startup(ByVal sender As Object, _
      ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) _
      Handles Me.Startup

            ' تغيير الثقافة الخاصة بالتطبيق عند بداية التطبيق
            ' قبل إظهار أية نافذة
            My.Application.ChangeCulture("Ar-Sy")

End Sub
```

و المحدد e يشير إلى الفئة Class StartupEventArgs التي توفر معلومات حول بداية التطبيق ولها عدة خصائص مثل Cancel التي تقوم بإلغاء الحدث و CommandLine التي تمرر محددات سطر الأوامر للتطبيق command-line arguments كما يمكنك الحصول على محددات سطر الأوامر أيضا باستخدام الأمر My.Application.CommandLineArgs من أي مكان في التطبيق

My.Application.Shutdown Event

حيث ينطلق هذا ا الحدث عند نهاية التطبيق فيمكنك استخدامه لمعالجة الأمور التي تحتاجها قبل نهاية التطبيق مباشرة

مثال :

```
Private Sub MyApplication_Shutdown(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Shutdown

    My.Application.Log.WriteEntry("Application Shut Down.")

End Sub
```

My.Application.StartupNextInstance Event

حيث ينطلق هذا ا الحدث عندما تحاول تشغيل نسخة ثانية من التطبيق المفترض أن يكون وحيد التواجد single-instance application والتطبيق مفتوح فعليا

مثال:

```
Private Sub MyApplication_StartupNextInstance(ByVal sender As Object, _
  ByVal e As
Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs)_
  Handles Me.StartupNextInstance

            Dim inputArgument As String = "/input="
            Dim inputName As String = ""
            For Each s As String In e.CommandLine
                If s.ToLower.StartsWith(inputArgument) Then
                    inputName = s.Remove(0, inputArgument.Length)
                End If
            Next
            If inputName = "" Then
                MsgBox("No input name")
            Else
                MsgBox("Input name: " & inputName)
            End If
```

```
End Sub
```

حيث يشير المحدد e للفئة StartupNextInstanceEventArgs التي تعيد معلومات حول تواجدات التطبيق Instancs Application حيث تحدد الخاصية BringToForeground فيما إذا كان يجب نقل التواجد الأول للتطبيق إلى الواجهة و الخاصية CommandLine تحدد محددات سطر الأوامر للتواجد الجديد للتطبيق

My.Application.UnhandledException Event

حيث ينطلق هذا ا الحدث عندما ينطلق استثناء Exception في تطبيقك لم تتم معالجته داخل التطبيق لتتمكن من إجراء معالجة لهذا الاستثناء مع ملاحظة أنه عندما يصل مسار تنفيذ البرنامج لهذه النقطة فإنه ينفذ هذا الإجراء ويخرج من البرنامج

مثال:

```
Private Sub MyApplication_UnhandledException(ByVal sender As Object, _
 ByVal e As _
Microsoft.VisualBasic.ApplicationServices.UnhandledExceptionEventArgs) _
  Handles Me.UnhandledException

           My.Application.Log.WriteException(e.Exception, _
      TraceEventType.Critical, "Unhandled Exception.")

End Sub
```

و المحدد e يشير إلى الفئة UnhandledExceptionEventArgs التي تحوي معلومات حول الخطأ مثل الخاصية Exception التي تحوي معلومات عن الخطأ و الخاصية ExitApplication التي تحدد فيما إذا كان يجب إنهاء التطبيق فور نهاية الإجرائية

NetworkChange.NetworkAvailabilityChanged Event

ينطلق هذا الحدث عندما تحدث تغييرات حول توافر الشبكة

```
Private Sub MyApplication_NetworkAvailabilityChanged(ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs) _
    Handles Me.NetworkAvailabilityChanged

           MsgBox(e.IsNetworkAvailable.ToString)

End Sub
```

حيث يشير المحدد e للفئة NetworkAvailableEventArgs التي تمرر معلومات حول توافر الشبكة ولها خاصية مفيدة هي IsNetworkAvailable التي تعيد قيمة منطقية هل الشبكة متوافرة أم لا

كيفية استخدام ملف التعريف الخاص بالتطبيق لاستهداف نسخة معينة من الفريموورك

يمكنك استخدام ملف التعريف الخاص بالتطبيق لتحديد ما هي نسخ الفريموورك التي يعتمد عليها التطبيق أو واحد أو أكثر من المكونات التي يستخدمها حيث يتوجب عليك تحديد رقم الإصدار ورقم البناء لكل من نسخ الفريموورك التي تريد دعمها باستخدام <supportedRuntime> أو <requiredRuntime> في ملف التعريف الخاص بالتطبيق

حيث يتم تحديد رقم نسخة الفريموورك التي يعتمد عليها التطبيق كما يلي:

- إذا كانت نسخة الفريموورك التي يعتمد عليها موجودة على الكمبيوتر الذي سيشغل التطبيق سيتم استخدامها من قبل التطبيق تلقائيا
- وإن لم تكن نسخة الفريموورك موجودة على ذلك الكمبيوتر ولم تقم بتحديد قيمة العنصر <supportedRuntime> فسيتم تشغيل التطبيق على أحدث نسخة من الفريموورك الموجودة على الجهاز
- وإن لم تكن نسخة الفريموورك موجودة على ذلك الكمبيوتر وملف التعريف الخاص بالتطبيق يحتوي على قيمة للعنصر <supportedRuntime> فالتطبيق سيتم تشغيله على أحدث نسخة من الفريموورك محددة في ملف التعريف الخاص بالتطبيق

كما يجدر الانتباه إلى أن ملف التعريف الخاص بالتطبيق يجب أن يكون له نفس اسم التطبيق ولكن يحمل اللاحقة . config فعلى سبيل المثال تطبيق يسمى MyExecutable.exe يجب أن يكون له ملف تعريف باسم MyExecutable.exe.config

يمكنك تعريف التطبيق ليعمل على نفس النسخة التي تم إنشاؤه عليها أو على نسخة أحدث فعلى سبيل المثال تطبيق تم إنشاؤه على الفريموورك 1.0 يمكنه العمل على أي من النسخ 1.0 أو 1.1 أو 2.0 أو عليها جميعا ولكن تطبيق تم إنشاؤه على الفريموورك 2.0 سيعمل فقط على الفريموورك 2.0

استهداف الفريموورك 1.1

قسم ملف التعريف المذكور في هذا القسم يوجه تطبيقا بني باستخدام الفريموورك 1.0 ليستخدم الفريموورك 1.1 وذلك في الحالات التالية:

- إذا كانت الفريموورك 1.1 موجودة

- إذا كانت كلتا نسختي الفريموورك 1.0 و 1.1 موجودتان

وإن كانت الفريموورك 1.0 فقط موجودة سيعمل البرنامج بسبب أن الفريموورك 1.0 لا تتعرف على العنصر <supportedRuntime> وسيستخدم النسخة الموجودة في ترويسة الملف التنفيذي للتطبيق

```
<?xml version ="1.0"?>
<configuration>
<startup>
<supportedRuntime version="v1.1.4322 / ">
</startup>
</configuration>
```

كيفية تحديد نسخة الفريموورك الصحيحة للمجمع

نستخدم <supportedRuntime> في ملف التعريف الذي يكون في العادة بصيغة xml حيث تحدد هذه القيمة ما هي نسخة الفريموورك التي يعمل عليها التطبيق وهي معتمدة في النسخة 1.1 وما بعد وتكون صيغتها

```
<supportedRuntime version="runtime version/">
```

و يكون version عبارة عن سلسلة نصية تحدد ما هي نسخة الفريمووورك التي يعتمد عليها التطبيق وهذه السلسلة النصية يجب أن تماثل اسم المجلد الموجود في المجلد الجذر للمجلد المنصب عليه الفريمووورك . وإذا لم يتم تحديد <supportedRuntime> فسيتم استخدام نفس رقم إصدار الفريمووورك الذي تمت ترجمة التطبيق عليه.

يتم استخدام <supportedRuntime> فقط في البرامج المبنية من خلال الفريمووورك 1.1 وما بعد أما بالنسبة لبرامج الفريمووورك 1.0 فيتم استخدام <requiredRuntime> عوضا عنها

والمثال التالي يوضح كيف يمكنك تحديد الاصدارات المدعومة من قبل التطبيق وذلك في ملف التعريف الخاص بالتطبيق

```
<configuration>
<startup>
<supportedRuntime version="v1.1.4322 / ">
<supportedRuntime version="v1.0.3705 / ">
</startup>
</configuration>
```

كيف نقوم بتوزيع مشروعنا باستخدام تقنية Click Once

كيف نقوم بخطوات النشر الأساسية للنشر على قرص CD أو DVD

١. افتح مشروعك ببيئة التطوير وقم بتجربته بالكامل وتأكد من خلوه من الأخطاء
٢. من قائمة Build اختر Configuration Manager ثم حدد Active Solution Configuration إلى Release إن كنت قد انتهيت تماما من تطوير المشروع وتريد نشره بالصورة النهائية
٣. من قائمة Build اختر Publish ProjectName حيث ProjectName هو اسم مشروعك عندها يفتح لك معالج النشر Publish Wizard
٤. في الصفحة الأولى من المعالج Where do you want to publish the application أدخل مسار المجلد الذي تريد نشر البرنامج إليه مثلا D:\My Program\Publish في المربع Specify the location to publish this application وإن لم يكن ذلك المجلد موجودا فسوف يتم إعلامك من أجل إنشائه الآن اضغط Next من أجل المتابعة والانتقال للصفحة التالية من المعالج
٥. في الصفحة How Will users install the application نقوم بتحديد الطريقة التي سيقوم بها المستخدمون بتنصيب البرنامج وبما أننا ننوي التوزيع باستخدام CD أو DVD فسوف نختار From CD-ROM or DVD-ROM ثم اضغط Next
٦. في الصفحة Where will the application check for updates سنختار The application will not check for updates ثم اضغط Finish
٧. وهنا سوف يتم ترجمة المشروع وتجهيز الملفات اللازمة للنشر في المجلد الذي قمت بتحديده في الخطوة الأولى وهنا يمكنك استخدام أي برنامج نسخ أقراص لنسخ محتويات مجلد النشر إلى قرص CD أو DVD

كيف يمكننا تضمين المكتبات الضرورية لتشغيل البرنامج مع ملفات التوزيع

١. من Solution Explorer افتح خصائص My Project ثم انتقل لصفحة Publish
٢. انقر زر Prerequisites فيفتح لك صندوق حوار Prerequisites
٣. اختر الخيار Download prerequisites from the same location as my application من أجل النشر على CD أو DVD أو يمكنك تحديد الخيار Download prerequisites from the component vendor's web site إذا كنت تنوي النشر على ويب عندها سيقوم برنامج الإعداد بالحصول على المكتبات من موقع الشركات الناشرة لها وبذلك تقلل من حجم الملفات التي سيتم رفعها على موقعك
٤. بشكل عام تكون بيئة التطوير قد اختارت المكتبات التي يعتمد عليها مشروعك مسبقا وإن أردت إضافة مكتبات أخرى من القائمة الموجودة في صندوق الحوار يكفي أن تضع إشارة داخل المربع بجانب تلك المكتبة لتضمينه ضمن مشروع النشر ثم اضغطOK
٥. ثم نقوم بنشر المشروع تماما كما ورد في بداية المقال

ملاحظات إضافية

- في صفحة Publish في خصائص My Project ستجد زر Application Files يمكنك من رؤية ملفات تطبيقك التي سيتم تضمينها عند النشر وستعرف أين سيتم تثبيتها من Publish Status فالقيمة Include ستكون في مجلد التطبيق و Data File ستكون في مجلد آخر سيتم إنشاؤه من أجل ملفات البيانات و Exclude لن يتم تضمين الملف عندما يتم توزيع البرنامج
- يمكنك تحديد بعض الخيارات الإضافية من زر Options في صفحة Publish في خصائص My Project فمثلا الخيار For CD installations automatically start setup when CD is inserted يسبب إنشاء ملف خاص ضمن الملفات في مجلد النشر يقوم بتشغيل برنامج الإعداد مباشرة عند وضع القرص في الحاسب مباشرة وخيارات أخرى مثل تحديد اسم الناشر واسم المنتج وغيرها
- إن قمت باختيار النشر إلى ويب في الصفحة الثانية من المعالج فعند الضغط على زر Next ستظهر لك صفحة تسألك عما إذا أردت أن يكون البرنامج متوفرا Online أو Offline أو الاثنين معا وكذلك الأمر عند اختيار الخيار From UNC path or file share

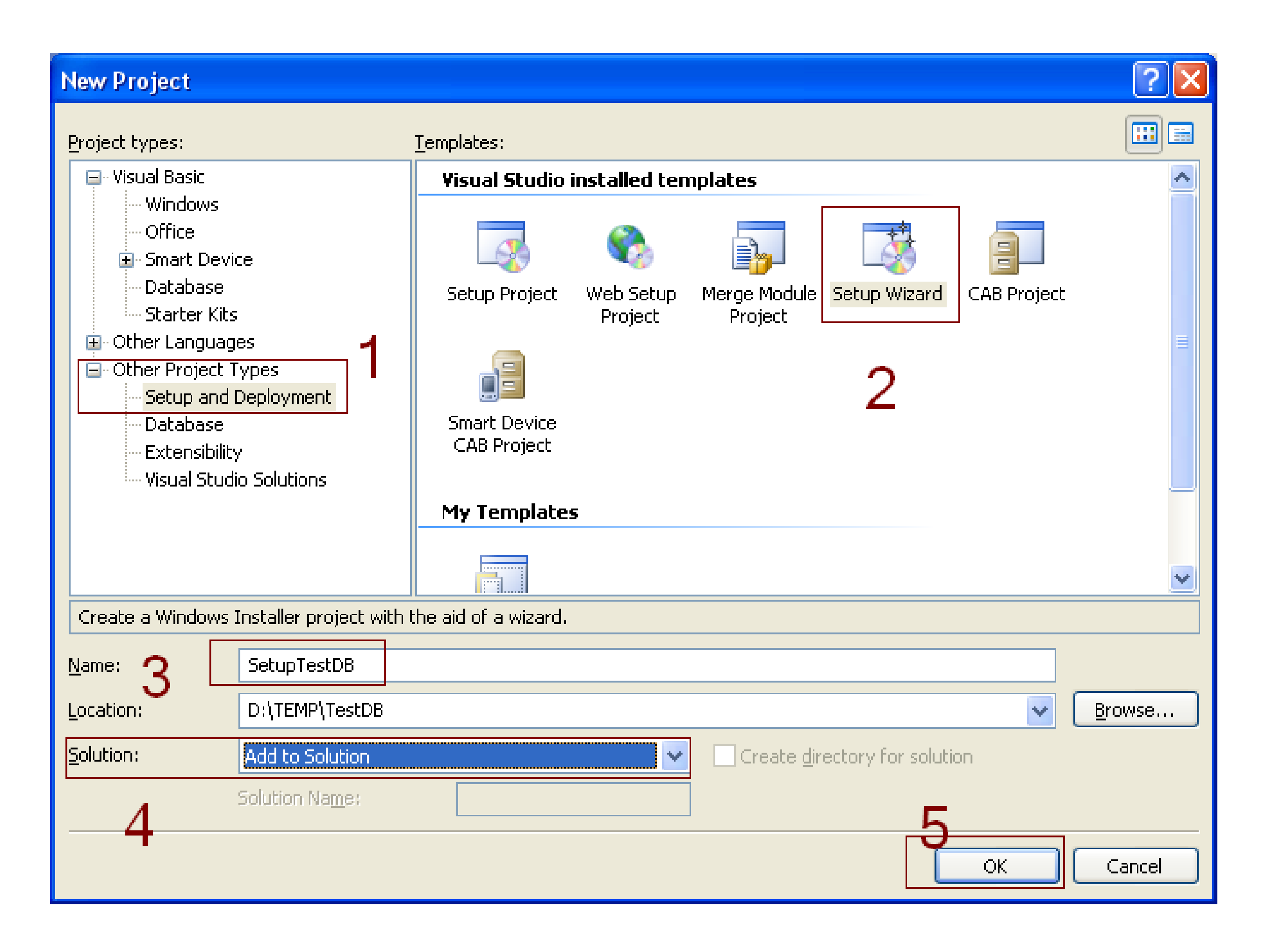
- عند النشر على ويب قد ترغب بتمكين مستخدميك من القيام بتحديث البرنامج تلقائيا ولتحديد هذه الخيارات عليك تحديد Publish Version من صفحة Publish في خصائص My Project ويفضل تحديد الخيار Automatically Increment revision with each publish حتى يتم تغيير رقم إصدار النشر كل مرة تقوم فيها بنشر مشروعك
- ومن أجل التحديثات التلقائية هناك زر Updates في صفحة Publish في خصائص My Project يمكنك من تحديد الخيارات الخاصة بعملية التحديث التلقائية لمشروعك ففي أعلى صندوق الحوار ستجد خيار تمكين التحديثات The Application should check for updates ثم تحدد هل سيقوم بالبحث عن التحديثات قبل أو بعد بدء التطبيق After the application starts أو Before the application starts وتحديد زمن وكيفية تكرار البحث عن التحديث في حالة اختيار After the Application starts وقد ترغب بتحديد رقم الإصدار الأدنى اللازم للتطبيق وخيار إضافي تقوم فيه بتحديد مسار موقع التحديث إن كان مختلفا عن مسار النشر
- من لا يجد عنده Configuration Manager افتح قائمة Tools ثم Options ثم Projects And Solutions وقم باختيار جميع الخيارات في القسم الأيمن والخيار المتعلق بهذه النقطة هو Show Advanced Build Configurations
- عند تنصيب البرنامج على جهاز العميل فإن قاعدة البيانات لا يتم وضعها في نفس المجلد مع التطبيق حيث يتم تنصيب التطبيق في مجلد خاص وقاعدة البيانات أو ملفات البيانات الأخرى في مجلد خاص بالبيانات ويمكن بعد التنصيب معرفة المجلد الذي تم تنصيب ملفات البيانات فيه باختبار القيمة My.Application.Deployment.DataDirectory مع الانتباه إلى استخدام حلقة Try لتصيد الخطأ في حال نشر البرنامج بطرق أخرى حيث لا تكون هذه القيمة معرفة

نشر مشروعك باستخدام SetupWizard

سنتحدث هنا عن طريقة SetupWizard في نشر مشروعك والشرح هنا ينطبق على النسخة Professional و نسخ أخرى ولكنه غير موجود في النسخة Express.

بعد الانتهاء من كتابة تطبيقك حان الوقت لعمل مشروع الإعداد لهذا التطبيق والحل المألوف منذ فيجول بايزيك 2003 هو عمل برنامج الإعداد عن طريق الـ SetupWizard وبالنسبة لأولئك الذين اعتادوا العمل بتلك الطريقة لن يجدوا سوى بعض الاختلافات البسيطة في طريقة العمل منها أن الـ Crystal Report لم تعد تتم إضافته عن طريق مكتبة دمج Merge Module بل أصبحت إضافته عن طريق ما يدعى بـ prerequisites ولم يعد هناك حاجة لوضع الرقم التسلسلي له

من قائمة File اختر new Project ثم اختر other project types ثم setup and deployment ثم اختر نوع المشروع SetupWizard ثم حدد اسم لمشروع الإعداد لديك ثم اضبط خيار solution إلى Add to solution ثم اضغط OK لبدء المعالج



اضغط next في الصفحة الأولى لنصل للصفحة الثانية التي هي البداية الحقيقية للمعالج وبما أننا نطور تطبيق ويندوز سنستخدم مبدئيا الخيارات الموضحة في الصورة.

حدد خياراتك كما يناسبك ثم انتقل للصفحة التالية التي يمكنك فيها تحديد نوع الخرج المناسب الذي تريد تضمينه في المشروع ثم اضغط next فتظهر لك صفحة خاصة بتضمين الملفات الإضافية التي ترغب بإضافتها للمشروع ثم عندما تنتهي يمكنك الضغط إلى زر next للانتقال للصفحة الأخيرة من المعالج والتي تعرض لك خياراتك التي حددتها من المعالج وبعد التأكد منها يمكنك الضغط على زر finish لإنهاء المعالج حيث يقوم بإنشاء مشروع الأعداد الخاص بك وسيقوم المعالج بعد انتهائه بفتح مستكشف نظام الملفات كبداية

Setup Wizard (2 of 5)

Choose a project type اختر نوع المشروع

The type of project determines where and how files will be installed on a target computer.

هل تريد انشاء برنامج إعداد لتنصيب تطبيقك

Do you want to create a setup program to install an application?

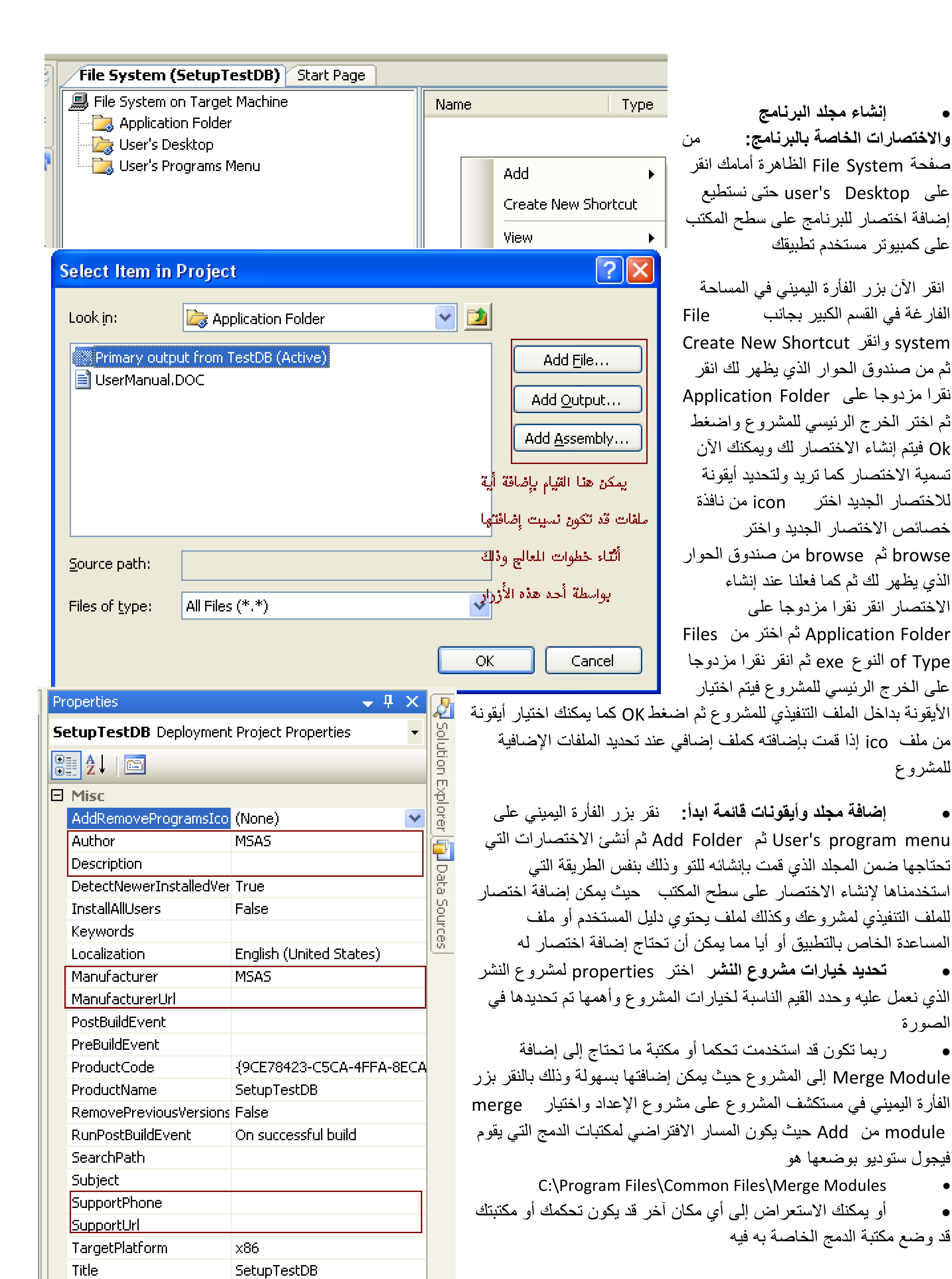
- (•) Create a setup for a Windows application إعداد لتطبيق ويندوز
- ( ) Create a setup for a web application إعداد لتطبيق ويب

Do you want to create a redistributable package? هل تريد إنشاء رزمة قابلة لإعادة التوزيع

- ( ) Create a merge module for Windows Installer مكتبة دمج لبرنامج إعداد ويندوز
- ( ) Create a downloadable CAB file ملف cab قابل للتنزيل

< Previous | Next > | Finish | Cancel



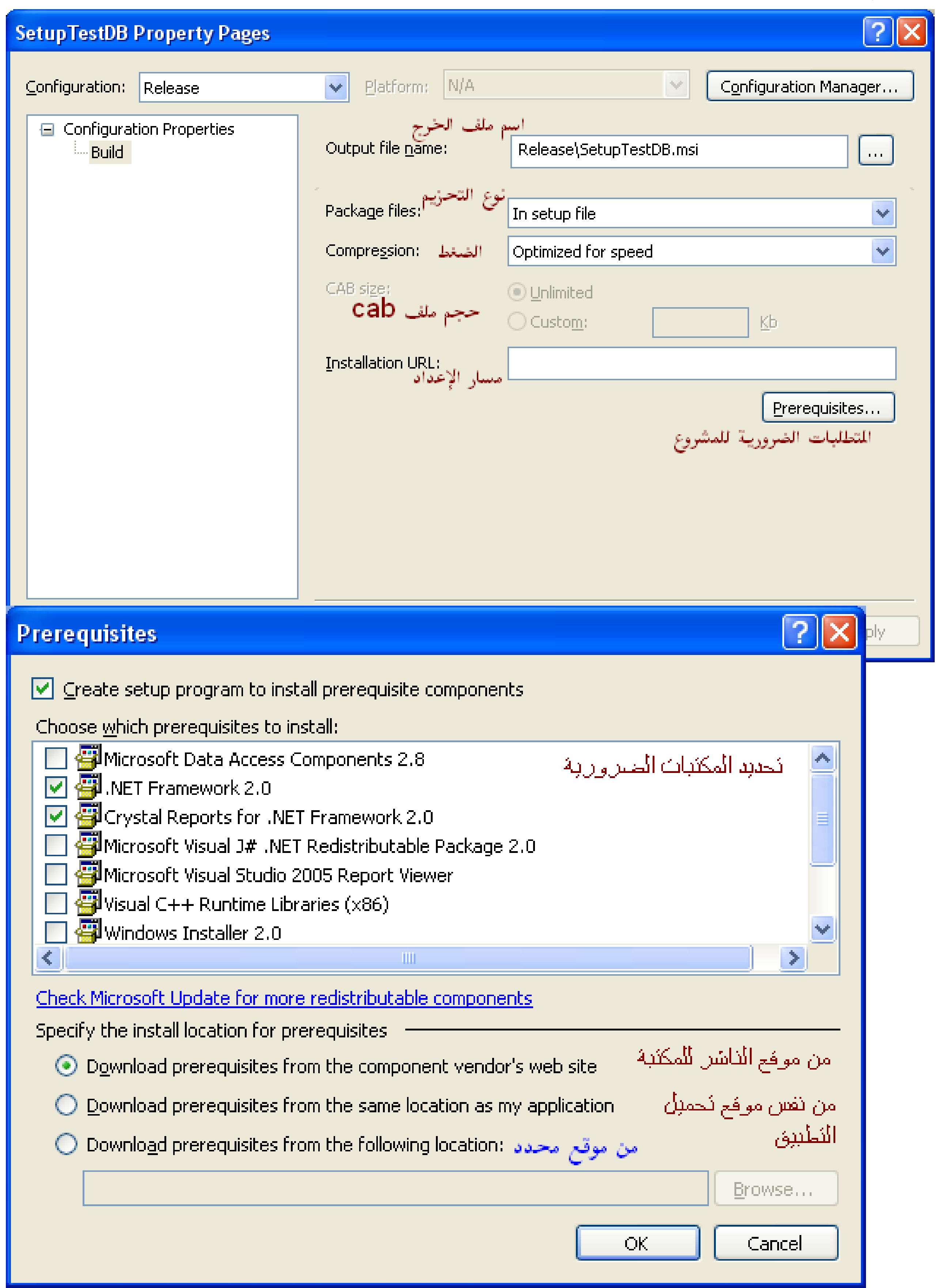


- **إنشاء مجلد البرنامج والاختصارات الخاصة بالبرنامج:** من صفحة File System الظاهرة أمامك انقر على user's Desktop حتى نستطيع إضافة اختصار للبرنامج على سطح المكتب على كمبيوتر مستخدم تطبيقك

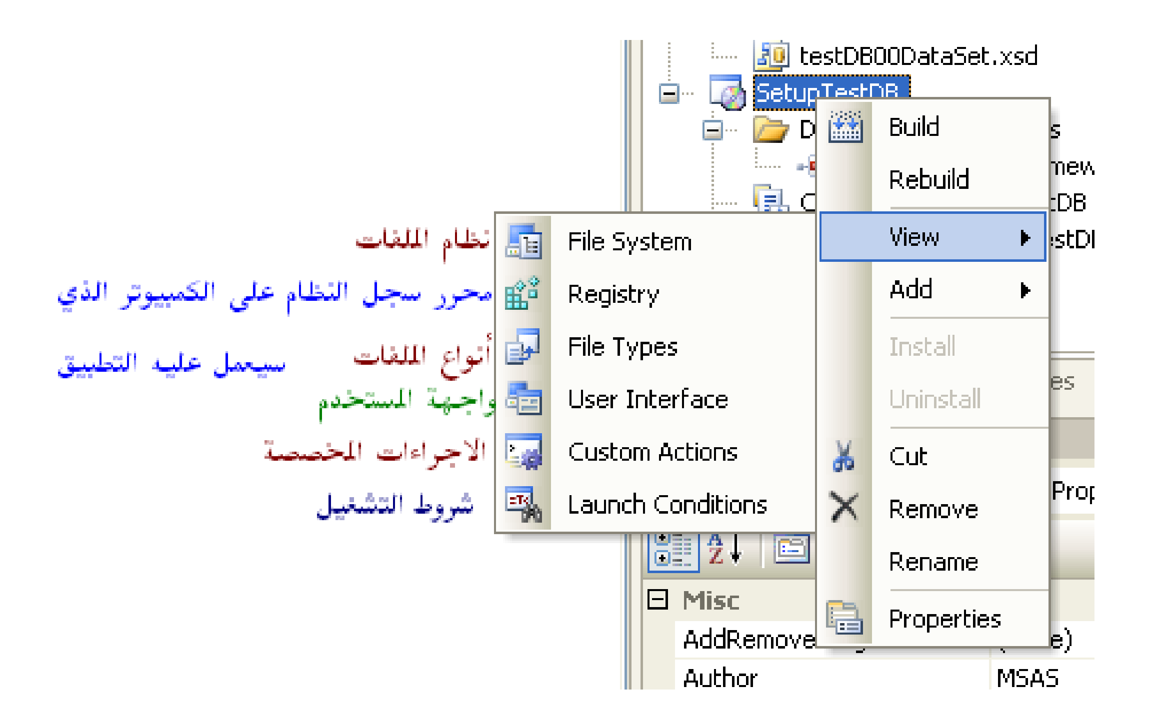
انقر الآن بزر الفأرة اليميني في المساحة الفارغة في القسم الكبير بجانب File system وانقر Create New Shortcut ثم من صندوق الحوار الذي يظهر لك انقر نقرا مزدوجا على Application Folder ثم اختر الخرج الرئيسي للمشروع واضغط Ok فيتم إنشاء الاختصار لك ويمكنك الآن تسمية الاختصار كما تريد ولتحديد أيقونة للاختصار الجديد اختر icon من نافذة خصائص الاختصار الجديد واختر browse ثم browse من صندوق الحوار الذي يظهر لك ثم كما فعلنا عند إنشاء الاختصار انقر نقرا مزدوجا على Application Folder ثم اختر من Files of Type النوع exe ثم انقر نقرا مزدوجا على الخرج الرئيسي للمشروع فيتم اختيار الأيقونة بداخل الملف التنفيذي للمشروع ثم اضغط OK كما يمكنك اختيار أيقونة من ملف ico إذا قمت بإضافته كملف إضافي عند تحديد الملفات الإضافية للمشروع

- **إضافة مجلد وأيقونات قائمة ابدأ:** نقر بزر الفأرة اليميني على User's program menu ثم Add Folder ثم أنشئ الاختصارات التي تحتاجها ضمن المجلد الذي قمت بإنشائه للتو وذلك بنفس الطريقة التي استخدمناها لإنشاء الاختصار على سطح المكتب حيث يمكن إضافة اختصار للملف التنفيذي لمشروعك وكذلك لملف يحتوي دليل المستخدم أو ملف المساعدة الخاص بالتطبيق أو أيا مما يمكن أن تحتاج إضافة اختصار له
- **تحديد خيارات مشروع النشر** اختر properties لمشروع النشر الذي نعمل عليه وحدد القيم الناسبة لخيارات المشروع وأهمها تم تحديدها في الصورة
- ربما تكون قد استخدمت تحكما أو مكتبة ما تحتاج إلى إضافة Merge Module إلى المشروع حيث يمكن إضافتها بسهولة وذلك بالنقر بزر الفأرة اليميني في مستكشف المشروع على مشروع الإعداد واختيار merge module من Add حيث يكون المسار الافتراضي لمكتبات الدمج التي يقوم فيجول ستوديو بوضعها هو
- C:\Program Files\Common Files\Merge Modules
- أو يمكنك الاستعراض إلى أي مكان آخر قد يكون تحكمك أو مكتبتك قد وضع مكتبة الدمج الخاصة به فيه

- الآن انقر بزر الفأرة اليميني على مشروع النشر الخاصة بك ثم اختر Properties ليظهر لك صندوق حوار يمكنك من تحديد خصائص إضافية لمشروع الإعداد الخاص بك وإذا أردت إضافة مكتبات ضرورية استخدمتها لمشروعك أو تحديد خيارات من أين سيحصل مستخدم تطبيقك على هذه المكتبات عندها اضغط على الزر prerequisites وقم بتحديد الخيارات التي تناسبك ثم اضغط Ok ثم Ok



تم شرح الخيارات الأساسية التي ستحتاجها كلما قمت بعمل مشروع إعداد ولكن هناك خيارات أخرى مفيدة قد تحتاجها أثناء عملك على مشروع الإعداد الخاص بك والآن وقد اطلعت على المهارات الأساسية لعمل برنامج الإعداد يمكنك بسهولة استخدام باقي هذه الخيارات بقليل من العمل فعلى سبيل المثال على الخيارات الأخرى انظر الصورة



القسم الحادي عشر - قواعد البيانات

ويضم المواضيع التالية:

- عندما تتصل بـ SQl Server 2005 تعاني من طول فترة الاتصال
- كيف تضيف إجراءاتك الخاصة إلى SQL Server
- كيف نقوم بالتبديل بين إصدارات سيكول سيرفر 2008 المختلفة
- إنشاء قاعدة بيانات أكسس من داخل فيجول بايزيك دوت نت

عندما تتصل بـ SQl Server 2005 تعاني من طول فترة الاتصال

الظاهرة

عندما تقوم بالاتصال بمحرك قواعد البيانات لـ Microsoft SQL Server 2005 ربما تعاني من وقت اتصال أطول مما اعتدت عليه في SQL Server 2000 و هذا يحدث عندما تتوافر لديك الشروط التالية:

- استخدام MDAC 2.8 أو أقدم
- البروتوكول TCP/IP غير مفعل في الـ SQL Server 2005
- بروتوكول الذاكرة المشتركة shared memory protocol مفعل في الكمبيوتر العميل وفي الكمبيوتر الذي يشغل SQL Server 2005

السبب

بروتوكول الذاكرة المشتركة shared memory protocol الذي يستخدم في MDAC غير متوافق مع بروتوكول الذاكرة المشتركة في SQL Server 2005 وعندما يحاول الكمبيوتر العميل الاتصال بـ SQL Server 2005 باستخدام بروتوكول الذاكرة المشتركة يقوم الكمبيوتر العميل مباشرة بالانتقال إلى البروتوكول TCP/IP من أجل القيام بالاتصال وبما أن البروتوكول TCP/IP غير مفعل يحصل التأخير عندها. حيث يقوم الكمبيوتر العميل بمحاولة الاتصال عبر البروتوكول TCP/IP لحوالي الثانية ثم يقوم بالاتصال عبر named pipe protocol و يوفر SQL Server 2005 اتصال named pipe protocol للعميل المحلي إذا كانت الذاكرة المشتركة مفعلة في SQL Server 2005

الحل

يمكنك استخدام إحدى الطرق التالية للتغلب على هذه المشكلة:

الطريقة الأولى

افتح عميل SQL Server Native Client للاتصال بـ SQL Server عوضا عن MDAC

الطريقة الثانية

قم بتفعيل البروتوكول TCP/IP في SQL Server 2005 وللقيام بذلك اتبع الخطوات التالية:

١. افتح SQL Server Configuration Manager
٢. وسع SQL Server 2005 Network Configuration
٣. انقر على Protocols for SQLInstanceName حيث SQLInstanceName هو اسم SQL Server 2005 لديك
٤. فعل البروتوكول TCP/IP

الطريقة الثالثة

على الكمبيوتر العميل أنشئ اسم مستعار Alias للكمبيوترات العميلة لـ SQL Server 2005 في SQL Server Configuration Manager ثم قم بتحديد named pipe protocol خاص لهذا الاسم المستعار:

١. افتح SQL Server Configuration Manager
٢. وسع SQL Native Client Configuration

٣. انقر على Alias ثم انقر New Alias
٤. في خيار البروتوكول Protocol حدد Named Pipes وأدخل المعلومات المناسبة لـ Alias Name و Pipe Name و Server

- نصيحة إضافية من التجربة قم بإيقاف تفعيل بروتوكول الذاكرة المشتركة shared memory protocol

كيف تضيف إجراءاتك الخاصة إلى SQL Server

سأقوم بشرح المثال هنا على الـ SQL Server Express حتى يستطيع الجميع المتابعة

أولا - هناك تعريفا صغيرا في SQL Server علينا عمله حتى يعمل المثال

- افتح من قائمة ابدأ من مجلد SQL Server البند SQL Server 2005 Surface Area Configuration ثم اختر Surface Area Configuration for Features ثم من صفحة CLR Integration اختر الخيار Enable CLR Integration ثم اختر OK ويفضل إعادة تشغيل الجهاز أو إعادة تشغيل SQL Server
- أنشئ مشروعا جديدا من نوع Class Library وسمه TestExtend إن كنت تريد المتابعة معي بنفس التسميات
- ملاحظة: سيكون الكود على VB .net ويمكن لمبرمجي C# المتابعة حيث سيكون العمل ذات نفسه بعد تحويل الأكواد البسيطة التي سنستخدمها هنا إلى C# مع الانتباه إلى أن Shared في VB تكافئ Static في C# عند تعريف الإجراء
- اجعل محرر الكود لديك والخاص بالمكتبة التي تقوم بإنشائها كالكود التالي

```
Imports System.Data.SqlTypes

Public Class TestClass

    Public Shared Function Triple(ByVal TrNum As SqlInt32) As SqlInt32
        Return TrNum * 3
    End Function

    Public Shared Function sFactorial(ByVal Tn As SqlDouble) As SqlDouble
        Dim Res As SqlDouble = 1
        For i As SqlDouble = Tn To 1 Step -1
            Res *= i
        Next
        Return Res
    End Function
End Class
```

يلاحظ هنا استعمال الكلمة Shared في تعريف الإجراءات التي سنستخدمها وذلك حتى نستطيع إضافتها لاحقا لقاعدة البيانات هنا نستخدم static functions فقط الآن قم بعمل Build للمشروع ثم قم بنسخ الملف TestExtend.dll الناتج إلى المجلد D:\ إذا كنت ترغب بالمتابعة معي حرفيا أو إلى أي مجلد آخر تريده مع مراعاة تعديل المسار في الأكواد التالية عند استيراد المكتبة لداخل قاعدة البيانات

- إذهب إلى محرر أوامر الدوس و شغل SQLCmd كما يلي

SQLCMD -S .\SQLEXPRESS

وعلى جهازي الأمر SQLcmd موجود في المسار

C:\Program Files\Microsoft SQL Server\90\Tools\Binn

بقية العمل سيكون من ضمن SQLCmd وهنا لافرق بين VB و C# انتقلنا الأن لأوامر SQL Server

- سنقوم بإنشاء قاعدة بيانات للتجربة - أدخل السطرين التاليين عند الموجه

create database SamerTest

GO

- ثم سنقوم بتغيير قاعدة البيانات المستخدمة إلى قاعدة البيانات الجديدة

use SamerTest

GO

- الآن سنقوم بإضافة المكتبة التي قمنا بإنشائها إلى قاعدة البيانات كالتالي مع ملاحظة الدقة في حالة الأحرف هنا حتى لو كنت استخدمت VB .net لإنشاء المكتبة

```
create assembly TestExtend

'from 'D:\TestExtend.dll

with PERMISSION_SET = SAFE

GO
```

حيث يقوم SQL Server بتخزين المكتبة TestExtend.dll ضمن قاعدة البيانات ولن يبقى حاجة لوجود مكتبتنا على القرص الصلب إلا على سبيل الاحتياط و نحدد الصلاحيات الخاصة بالمكتبة بأنها آمنة بواسطة الخيار PERMISSION_SET = SAFE

- الآن سنقوم باستيراد الإّجراءات الموجودة ضمن المكتبة إلى SQL Server حتى نستطيع استخدامها

```
create function sFactorial

 (@Tn Float) RETURNS Float

as external name TestExtend.[TestExtend.TestClass].sFactorial

go


create function Triple

 (@tn int) returns int

as external name TestExtend.[TestExtend.TestClass].Triple

go
```

الكود يقوم بإنشاء إجراء يستخدم كود الإجراء الموجود في المكتبة حيث نعرف اسم الاجراء create function sFactorial ثم نقوم بتحديد محددات الدخل بالضبط كما عرفناها في مكتبتنا @tn float ثم نحدد نوع الخرج returns float ثم نحدد مسار الإجراء في مكتبتنا as external name TestExtend.[TestExtend.TestClass].sFactorial

- الآن وبعد إضافتنا المكتبة والإجراءات لقاعدة البيانات بقي أن نقوم بتجريبها - استخدم الكود التالي لبناء الجداول التجريبية

```
create table Test(Tn int)

GO

insert into acctest values(15)

insert into acctest values(25)

insert into acctest values(50)

insert into acctest values(40)

insert into acctest values(70)

GO
```

create table AccTest(Amount int,Interest int, Days Int)

GO

insert into acctest values(150000,5,95)

insert into acctest values(250000,5,95)

insert into acctest values(50000,4,105)

insert into acctest values(2500000,3,180)

insert into acctest values(25000,7,73)

GO

- استخدم استعلامات شبيهة بالتالي وانظر كيف تعمل إجراءاتنا ضمن علاقة Select تماما كالإجراءات الموجودة سابقا ضمن SQL Server

SELECT Tn, DBO.Triple(Tn) from Test

GO

SELECT Tn, DBO.Triple(Tn), sFactorial(Tn) from Test

GO

لا تغلق SQLCmd مازلنا بحاجته

- الآن ارجع وافتح مشروع المكتبة التي أنشأناها سابقا وأضف إليها الإجراء التالي ثم اعمل Build للمكتبة بعد إضافة الإجراء

```
Public Shared Function sInterest(ByVal Value As SqlInt32, _
    ByVal Interest As SqlInt32, ByVal Days As SqlInt32) As SqlInt32

    Return CType(((Value * Interest * Days) / 36500), SqlInt32)

End Function
```

- استخدم ملف المكتبة المحدث لتحديث النسخة الموجودة ضمن قاعدة البيانات كالتالي وذلك من ضمن SQLCmd

alter assembly TestExtend

'from 'D:\TestExtend.dll

with PERMISSION_SET = SAFE

GO

حيث يقوم الكود السابق بتحميل النسخة الجديدة من المكتبة مكان النسخة القديمة في قاعدة البيانات

- الآن أضف الإجراء الجديد حتى نستخدمه

create function sInterest

(@Value int, @Interest int, @days int) returns int

as external name TestExtend.[TestExtend.TestClass].sInterest

go

- استخدم استعلاما شبيها بالتالي لتجربة الإجراء الجديد

select amount, dbo.sInterest(amount,interest,days) as DueInterest,

dbo.Triple(dbo.sInterest(amount,interest,days)) as TripleDue

from acctest

go

- إذا أردت حذف إجراء يمكنك ذلك ببساطة

DROP FUNCTION sInterest

GO

- وإذا أردت حذف المكتبة بكاملها نقوم أولا بحذف جميع الإجراءات المرتبطة بها ثم نستخدم الكود

DROP ASSEMBLY TestExtend

GO

- وإذا أردت استخدام تلك الإجراءات من داخل كود برنامجك: أضف قاعدة البيانات لمشروعك وقم بإنشاء الاتصال و الـ Dtatset بالطرق الاعتيادية - تأكد من استيراد Functions ضمن الـ Dataset
- في محرر الكود أضف الاستيراد التالي قبل كل شئ

```
Imports System.Data.SqlTypes
```

- استخدم إجراء شبيها بالتالي للاستعلام ضمن الكود وذلك بافتراض أن النموذج يحوي ListBox عدد اثنان وزر أوامر يحوي الكود التالي وذلك باستخدام الطريقة الاعتيادية لتنفيذ أمر Select من ضمن الكود حيث يمكنك ملاحظة استخدام أوامر تعديل صلاحيات المستخدم الحالي على SQL Server قبل تنفيذ جملة الاستعلام

```
Me.ListBox1.Items.Clear()
Me.ListBox2.Items.Clear()

Try
    Using MoCon As New SqlConnection(My.Settings.TestConnectionString)
        Dim MoCmd As New SqlCommand
        Dim MoReader As SqlDataReader

        MoCmd.CommandText = "sp_configure 'clr enabled', 1;"

        MoCmd.CommandType = CommandType.Text
        MoCmd.Connection = MoCon

        MoCmd.Connection.Open()

        MoCmd.ExecuteNonQuery()

        MoCmd.CommandText = "RECONFIGURE;"
        MoCmd.ExecuteNonQuery()

        MoCmd.CommandText = "SELECT Tn, DBO.TRIPLE(Tn) AS TRIP FROM TEST"

        MoReader = MoCmd.ExecuteReader

        If MoReader.HasRows = True Then
            While MoReader.Read
```

```
                Me.ListBox1.Items.Add(MoReader.Item("Tn").ToString)
                Me.ListBox2.Items.Add(MoReader.Item("TRIP").ToString)
            End While
        End If
        MoCmd.Connection.Close()
    End Using
Catch ex As Exception
    MsgBox(ex.ToString)
End Try
```

كيف نقوم بالتبديل بين إصدارات سيكول سيرفر 2008 المختلفة

في الحقيقة توجد خطوة إضافية تحتاج لعملها عند قيامك بالترقية من نسخة سيكول سيرفر 2008 إلى أخرى فمثلا عندما تقوم بالترقية من نسخة Express إلى نسخة Express With Tools تدعى هذه الخطوة الإضافية بترقية الإصدار Edition Upgrade وأحيانا SKU Upgrade وبالنتيجة ستحتاج لتشغيل معالج التنصيب مرتين عندما تريد الترقية من نسخة موجودة إلى نسخة مختلفة

لماذا نحتاج لعمل ذلك

النقطة الأساسية هي أن شجرة المزايا التي تراها مبنية على النسخة الموجودة حاليا في جهازك وعندما تريد الترقية من نسخة إلى أخرى تكون المعلومات الموجودة على الجهاز متعلقة بالإصدارة القديمة ونتيجة لهذا عليك تحديث تلك المعلومات لتتضمن الإصدارة الجديدة قبل أن تستطيع إضافة الخصائص الجديدة المتعلقة بالنسخة الجديدة.

وتكون عملية ترقية النسخة العامة حدث نادر ولكنها تحدث بشكل أكبر بالنسبة لمستخدمي الإصدارات Express بما انه على الأغلب تكون قد نصبت منتجا آخر مثل فيجول ستوديو الذي يكون متضمنا على النسخة الأساسية فقط وتريد إضافة ميزة من النسخ الأخرى الأكثر اكتمالا مثل Express with tools

ماذا علينا أن نفعل

ترقية الإصدار Edition Upgrade هي في الحقيقة مسيرة أخرى عبر معالج التنصيب التي تقوم فيها بإدخال مفتاح المنتج الخاص بالنسخة الجديدة عندما تنتقل إلى النسخة المدفوعة وبما أن جميع النسخ Express تمتلك مفتاح منتج مدخل مسبقا تكون هذه الخطوة حقيقة هي تنزيل تلك الحزمة إلى قرصك الصلب والمرور عبرها مرتين.

في المرة الأولى عندما تقوم بتشغيل الحزمة عليك أن تختار خيار الترقية إلى نسخة مختلفة upgrade to a different edition وهذا سيمر عبر مجموعة من الفحوصات وتعطيك الخيار لإدخال مفتاح المنتج الجديد وفي حالة النسخ Express لن تضطر لإدخال مفتاح المنتج وحالما تنتهي من ترقية النسخة عليك تشغيل معالج التنصيب مرة أخرى وهذه المرة عليك اختيار إضافة مزايا لنسخة موجودة Add Features to an existing Instance وستمر عبر عدة صفحات معالج حتى تصل لشجرة المزايا التي تعرض لك قائمة بالمزايا المتوفرة مع الإصدارة الجديدة

هناك دوما استثناء ما هو

نعم هناك دائما استثناء ففي حالة أمكنك تخطي خطوة ترقية الإصدار بين نسخة SQL Express With Tools و النسخة SQL Express With Advanced Services إذا كنت قد قمت بتنصيب النسخة with tools سلفا وتريد إضافة مزايا من النسخة Advanced Services يمكنك الذهاب مباشرة إلى إضافة مزايا Add Feature وتخطي ترقية الإصدار.

راجيا أن تكون هذه السطور قد أوضحت الأمور عندما تريد ترقية نسختك

القسم الثاني عشر - التعابير النظامية

ويضم المواضيع التالية:

- البحث عن الكلمات والنصوص المقتبسة
- التحقق من السلاسل النصية والأرقام والتواريخ
- تعابير نظامية شائعة جاهزة للاستخدام

البحث عن الكلمات والنصوص المقتبسة

إحدى العمليات الشائعة للتعابير النظامية Regular Expressions هي تقسيم سلسلة نصية طويلة إلى كلمات وهي تعتبر أبسط عملية يمكن أن تقوم بها باستخدام التعابير النظامية

```
Dim text As String = "A word with àccéntèd vowels, and the 123 number."
Dim pattern As String
pattern = "\w+"
For Each m As Match In Regex.Matches(text, pattern)
    Console.WriteLine(m.Value)
Next
```

والمشكلة في هذا المثال المبسط في انه يحتوي على أرقام وشرطات في مجموعة Collection النتائج وقد لا ترغب بذلك وستكون المحاولة الأفضل على الشكل

```
pattern = "\w+"
```

وهذا سيعمل بشكل أفضل ولكنه سيفشل بإضافة الكلمات كاملة إذا كانت تحتوي على حروف مشددة accented characters أو حروف من لغات أخرى كاليونانية مثلا وتحت النسخ السابقة من الفريموورك يمكنك أن تحل هذه المشكلة باستخدام \p والذي يمكنك من استخدام محارف يونيكود فعلى سبيل المثال \p{Ll} يمثل أي محرف صغير بينما \p{Lu} يمثل أي محرف كبير وبذلك يكون حل المشكلة كالتالي

```
pattern = "(\p{Lu}|\p{Ll})+"
```

تم تقديم ميزة طرح فئات المحارف في الفريموورك 2.0 لتقدم حلا جديدا لهذه المشكلة معتمدا على حقيقة أنه يمكنك طرح الأرقام والشرطات من مجال المحارف المعبر عنها بواسطة \w

```
pattern = "[\w-[0-9_]]+"
```

وعندما تستخرج الكلمات غالبا ما تريد اسقاط كلمات الضوضاء مثل The و a و an وما مثل ذلك حيث يمكنك اسقاط هذه الكلمات ضمن حلقة For ... Each ولكن الأكثر أناقة هو ترك التعابير النظامية لتقوم بالتخلص منهم

```
pattern = "\b(?!(the|a|an|and|or|on|of|with)\b)\w+"
text = "A fox and another animal on the lawn"
For Each m As Match In Regex.Matches(text, pattern, RegexOptions.IgnoreCase)
    Console.Write("{0} ", m.Value) ' => fox another animal lawn
Next
1
```

التعبير \w في المثال السابق يحدد أننا نبحث عن كلمة بينما التعبير (?!...]b) يحدد أن النتيجة يجب أن لا تحتوي أحد كلمات الضوضاء وبذلك تكون النتيجة بأناقة هي أن النموذج pattern سيطابق جميع الكلمات عدا تلك الموجودة في قائمة الضوضاء.

مشكلة أخرى شائعة وهي عندما تريد أن تعتبر نص مقتبس Quoted ككلمة واحدة وذلك كمثل عندما تقوم بمعالجة أمر يتم تمريره عبر سطر الأوامر والمثال التالي يستخدم تعبيرا نظاميا يطابق كلمة مفردة أو نص مضمن ضمن علامات اقتباس مفردة أو مزدوجة

```
' For simplicity's sake, use \w+ to match an individual word.
pattern = "(?<q>[""']).*?\k<q>|\w+"
```

لاحظ أن .*؟ تقوم بعمل مطابقة كسولة بحيث تطابق أي محرف بين علامات الاقتباس بينما لا تطابق علامات الاقتباس للإغلاق

ربما ترغب أحيانا باستخراج كلمات فريدة مثلا عمدما تريد عمل قاموس بجميع الكلمات الموجودة في ملف نصي حيث يمكن أن يشكل جدول هاش Hashtable حلا لتذكر الكلمات التي تم إيجادها حتى الآن

```
Dim text As String = "one two three two zone four three"
Dim re As New Regex("\w+")
Dim words As New Hashtable()
For Each m As Match In re.Matches(text)
    If Not words.Contains(m.Value) Then
        Console.Write("{0} ", m.Value)
        words.Add(m.Value, Nothing)
    End If
Next
```

وبشكل آخر يمكنك تحقيق ذلك باستخدام التعبيرات النظامية

```
pattern = "(?<word>\b\w+\b)(?!.+\b\k<word>\b)"
For Each m As Match In Regex.Matches(text, pattern)
    Console.Write(m.Value & " ")
Next
```

التعبير (?<word>]b]w+]b) يطابق سلسلة محارف وأرقام (\w) على حدود الكلمة (\b) وتحدد في هذا السياق الاسم "word" والتركيب (?!) يعني أن الكلمة يجب أن لاتطابق كلمة تم إيجادها سابقا (المرجع الخلفي \k<word>) وحتى إذا كانت هناك محارف أخرى في المنتصف ممثلة بالتسلسل +. وبتعبير واضح يعني التعبير النظامي طابق أي كلمة في النص بحيث أن لاتكون متبوعة بورود آخر نفس الكلمة أو ببساطة أكثر احصل على الكلمات التي لها ورود واحد في الوثيقة أو آخر ورود للكلمة المكررة وبهذا سيتم إيجاد كافة الكلمات الفريدة بصورة صحيحة

لاحظ أن المحارف \b في التعبيرات النظامية تمنع المطابقات الجزئية فـ one لاتطابق zone وبتعبير نظامي مختلف قليلا يمكنك إيجاد الكلمات المكررة في الوثيقة

```
pattern = "(?<word>\b\w+\b)(?=.+\b\k<word>\b)"
```

حيث أن (?=) يعني أن الكلمة المطابقة يجب أن تكون متبوعة بورود آخر لها ومع أن تقنيات التعبيرات النظامية أنيقة فإن التعبير (?=) الخاص بالنظر للأمام يجعلها غير كفؤة نسبيا فمثلا عند معالجة مصدر نصي يحتوي على حوالي مليون حرف سيكون استخدام التعبيرات النظامية أبطأ بحوالي 8 مرات من التقنية التي تستخدم جدول هاش hashtable مساعد لكي يحتفظ بسجل عن الكلمات التي تم إيجادها سابقا

نوع أخير من البحث عن الكلمات يمكن أن نتحدث عنه هنا هو البحث التقريبي وذلك عندما تبحث عن نصين يجب إيجادهما قريبين من بعضهما في النص المصدر بحيث لايكون أكثر من عدد N من الكلمات يفصل بينهما فعلى سبيل المثال النص "one two three two zone four three" هو المصدر فالبحث التقريبي للكلمات "one" و "four" والعدد N مساوي 4 سيتم بنجاح بينما سوف يفشل إذا كان العدد N مساوي لـ 3 ويمكن أن يكون نموذج البحث ببساطة كالمثال

```
pattern = "\bone(\W+\w+){0,4}\W+\bfour\b"
If Regex.IsMatch(text, pattern, RegexOptions.IgnoreCase) Then
    ' At least one occurrence of the words "one" and "four"
    ' with four or fewer words between them.
End If
```

ويمكن أن تحدد دالة function تأخذ سلسلة نصية كمدخل وكلمتان ورقم يمثل المسافة الأعظمية بينهما وتعيد خرج بشكل مجموعة مطابقات MatchCollection

```
Function ProximityMatches(ByVal text As String, ByVal word1 As String, _
   ByVal word2 As String, ByVal maxDistance As Integer) As MatchCollection

    Dim pattern As String = "\b" & word1 & "(\W+\w+){0," &
maxDistance.ToString() _
       & "}\" & "W+\b" & word2 & "\b"
    Dim re As New Regex(pattern, RegexOptions.IgnoreCase)
    Return re.Matches(text)
End Function
```

وبهذا تصبح قطعة الكود السابقة بالشكل

```
Dim mc As MatchCollection = ProximityMatches(text, "one", "four", 4)
If mc.Count > 0 Then
    …
End If
```

التحقق من السلاسل النصية والأرقام والتواريخ

يمكننا استخدام نموذج بحث كنموذج تحقق وذلك بتضمينه ضمن الرموز ^ و $ واستخدام الطريقة IsMatch بدلا من الطريقة Matches فمثلا يتحقق الكود التالي من أن النص يحتوي على خمسة أرقام تمثل رمز منطقة في أمريكا

```
pattern = "^\d{5}$"
If Regex.IsMatch(Text, pattern) Then
    ' It's a string containing five digits.
End If
```

وهنا ستصبح الأمور أكثر إثارة عندما تريد استبعاد بعض التراكيب من مجموعة النصوص الصحيحة الشئ الذي يمكنك فعله بواسطة التركيب (?!) فعلى سبيل المثال التسلسل 00000 لا يعتبر رمز منطقة صالح حيث يمكنك استبعاده كالتالي

```
pattern = "^(?!00000)\d{5}$"
```

يمكنك استخدام تراكيب النظر مقدما (?=) للتأكد من أن نص الإدخال يحتوي على جميع المحارف ضمن فئة معطاة بغض النظر عن مكانها فمثلا يمكنك استخدام التركيب التالي لفرض سياسة كلمة سر قوية والتأكد من أن مستخدم البرنامج يقوم بإدخال ثمانية محارف كحد أدنى وأنها تحتوي على أرقام وحروف كبيرة وصغيرة

```
pattern = "^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])\w{8,}$"
```

دعنا نرى كيف يعمل هذا التركيب. الشرط الأول (?=.*\d) يجعل البحث يفشل منذ البداية إذا كان جزء النص على يمينه لا يحتوي على أرقام والتركيب (?=.*[a-z]) يتأكد أن النص المدخل يحتوي على حروف صغيرة وبالمثل التركيب (?=.*[A-Z]) يتأكد من احتواء النص على حروف كبيرة. وتراكيب النظر للأمام هذه لا تستهلك أية محارف وبهذا يتأكد التركيب \w{8,} المتبقي من أن النص يحتوي على ثمانية محارف كحد أدنى

ويظهر في التحقق من رقم في مجال معطى مشاكل مثيرة وبشكل عام قد تريد استخدام التعابير النظامية للتحقق من الأرقام والتواريخ لأن النوع DateTime يزودك بالطرق Parse و Try-Parse وجميع الأنواع الرقمية أيضا مما يوفر مرونة أكثر ومع ذلك في بعض الحالات تكون التعابير النظامية قابلة للتطبيق حتى في هذه المهمة فمثلا قد تريد استخراج الأرقام والتواريخ الصحيحة من وثيقة أطول والتأكد من أن العدد الصحيح Integer يملك قيمة حتى حد معين تعتبر مشكلة عادية بالطبع

```
' Validate an integer in the range of 0 to 9,999; accept leading zeros.
pattern = "^\d{1,4}$"
```

تركيب النظر للأمام (?!) السلبي يجعلك تتحكم بعدة حالات

```
' Validate an integer in the range 1 to 9,999; reject leading zeros.
pattern = "^(?!0)\d{1,4}$"
…
' Validate an integer in the range 0 to 9,999; reject leading zeros.
' (Same as previous one, but accept a single zero as a special case.)
pattern = "^(0|(?!0)\d{1,4})$"
…
```

إذا كان الحد الأدنى للمجال المقبول ليس بالصيغة 999...99 تبقى لديك إمكانية استعمال التعابير النظامية لعمل التحقق ولكن التركيب يصبح أكثر تعقيدا فمثلا يتحقق التركيب التالي من أن الرقم ضمن المجال من 0 إلى 255 بدون أصفار سابقة

```
pattern = "^(25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)$"
```

التركيب [0-5]25 يتحقق من الأرقام في المجال من 250 إلى 255 والتركيب 2[0-4]\d يتحقق من الأرقام ضمن المجال من 200 إلى 249 والتركيب 1\d\d يتحقق من الأرقام ضمن المجال من 100 إلى 199 والتركيب [1-9]\d يهتم بالأرقام من 10 إلى 99 وأخيرا التركيب \d يغطي المجال من 0 إلى 9 وبتعديل بسيط على التركيب يمكنك التحقق من رقم IP مكون من أربعة أقسام مثل 192.168.0.11

```
pattern = "^((25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)\.){3}" _
    & "(25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)$"
```

وتصبح الأمور أكثر تعقيد بسرعة ضمن المجالات الأكبر

```
' Validate an integer number in the range 0 to 65,535; leading zeros are OK.
pattern = "^([1-5]\d{4}|6[0-4]\d{3}|65[0-4]\d{2}|655[0-2]\d|6553[0-4]|\d{1,4})$"
```

وقد يكون للأرقام إشارة تسبقها تحتاج لمعالجة خاصة

```
' Validate an integer in the range -32,768 to 32,767; leading zeros are OK.
pattern = "^(-?[12]\d{4}|-?3[0-1]\d{3}|-?32[0-6]\d{2}|-?327[0-5]\d|" _
            & "-?3276[0-7]|-32768|-?\d{1,4})$"
```

لاحظ في التركيب السابق أن الحالة الخاصة 3278- يجب التعامل معها بشكل مستقل وكل التركيب الباقي يملك إشارة ناقص اختيارية سابقة ويمكنك استخدام تقنية شبيهة للتحقق من قيمة وقت

```
' Validate a time value in the format hh:mm; the hour number can have a leading
zero.
pattern = "^(2[0-3]|[01]\d|\d):[0-5]\d$"
```

والتحقق من قيم التاريخ قد تكون أكثر صعوبة بسبب احتواء كل شهر على عدد مختلف من الأيام وفوق كل ذلك يعتمد اليوم الصحيح في شهر شباط على كون السنة كبيسة أم لا وقبل تقديم حل لهذه المشكلة دعنا نرى كيف يمكن للتعابير النظامية التحقق من أن أي رقم معطى ذو خانتين يقبل القسمة على 4

```
' If the first digit is even, the second digit must be 0, 4, or 8.
' If the first digit is odd, the second digit must be 2 or 6.
pattern = "^([02468][048]|[13579][26])$"
```

وباستخدام افتراض مبسط بأن رقم السنة ذو خانتين فقط فسيكون بالتالي التاريخ المعطى في القرن الحالي مما سيمكننا من تبسيط التعبير النظامي بشكل واضح لأن السنة2000 كبيسة بعكس السنة1900 و 2100 ولتوضيح التعبير النهائي قسم التركيب إلى أربعة سطور

```
' This portion deals with months with 31 days.
Dim p1 As String = "(0?[13578]|10|12)/(3[01]|[012]\d|\d)/\d{2}"
' This portion deals with months with 30 days.
Dim p2 As String = "(0?[469]|11)/(30|[012]\d|\d)/\d{2}"
' This portion deals with February 29 in leap years.
Dim p3 As String = "(0?2)/29/([02468][048]|[13579][26])"
' This portion deals with other days in February.
Dim p4 As String = "(0?2)/(2[0-8]|[01]\d|\d)/\d{2}"
' Put all the patterns together.
pattern = String.Format("^({0}|{1}|{2}|{3})$", p1, p2, p3, p4)
' Check the date.
If Regex.IsMatch(Text, pattern) Then
    ' Date is valid.
End If
```

وإن كان رقم السنة ذو خانتين أو أربع خانات وجب علينا الأخذ بعين الاعتبار أنه ليست جميع السنوات القابلة للقسمة على 100 كبيسة إلا إن كانت قابلة للقسمة على 400 مما يجعل التركيب النظامي أكثر تعقيدا ولكنك أصبحت تملك الخبرة الكافية لفهم كيف يعمل الكود التالي

```
' This portion deals with months with 31 days.
Const s1 As String = "(0?[13578]|10|12)/(3[01]|[12]\d|0?[1-9])/(\d\d)?\d\d"
' This portion deals with months with 30 days.
Const s2 As String = "(0?[469]|11)/(30|[12]\d|0?[1-9])/(\d\d)?\d\d "
' This portion deals with days 1-28 in February in all years.
Const s3 As String = "(0?2)/(2[0-8]|[01]\d|0?[1-9])/(\d\d)?\d\d"
' This portion deals with February 29 in years divisible by 400.
Const s4 As String = "(0?2)/29/(1600|2000|2400|2800|00)"
' This portion deals with February 29 in noncentury leap years.
Const s5 As String = "(0?2)/29/(\d\d)?(0[48]|[2468][048]|[13579][26])"
' Put all the patterns together.
pattern = String.Format("^({0}|{1}|{2}|{3}|{4})$", s1, s2, s3, s4, s5)
```

وسيكون من السهل عمل تعبير نظامي للتواريخ من الشكل dd/mm/yy وذلك بالأخذ بعين الاعتبار اختلاف الحرف الفاصل بين أقسام التاريخ.

تعابير نظامية شائعة جاهزة للاستخدام

<table>
<tr><th>Pattern</th><th>الوصف</th></tr>
<tr><td>\d+</td><td>عدد صحيح موجب</td></tr>
<tr><td>[+-]?\d+</td><td>عدد صحيح موجب أو سالب بإشارة اختيارية</td></tr>
<tr><td>[+-]?\d+(\.\d+)?</td><td>عدد بنقطة عائمة بإشارة وقسم عشري اختياريين</td></tr>
<tr><td>[+-]?\d+(\.\d+)?(E[+-]?\d+)?</td><td>عدد بنقطة عائمة يمكن تمثيله اختياريا بالشكل الأسي</td></tr>
<tr><td>[0-9A-Fa-f]+</td><td>رقم ست عشري</td></tr>
<tr><td>\w+</td><td>سلسلة من الحروف والأرقام والشرطات</td></tr>
<tr><td>[A–Z]+</td><td>كلمة ذات محارف كبيرة</td></tr>
<tr><td>[A–Z][a–z]+</td><td>كلمة تكون فيها الحرف الأول كبير والباقي صغير</td></tr>
<tr><td>[A–Z][A–Za–z']+</td><td>كلمة فيها الحرف الأول كبير والبقية يمكن أن يكون فيها حروف كبيرة وفواصل علوية</td></tr>
<tr><td>[A–Za–z]{1,10}</td><td>كلمة بعشر حروف أو أقل</td></tr>
<tr><td>[A–Za–z]{11,}</td><td>كلمة بأحد عشر حرفا أو أكثر</td></tr>
<tr><td>[A-Za–z_]\w*</td><td>متغير فيجول بايزيك أو سي شارب يبدأ بحرف أو شرطة واختياريا تتمته بحروف أو أرقام أو شرطات</td></tr>
<tr><td>(?<q>["'])*?\k<q></td><td>نص مقتبس ضمن علامتي اقتباس مفردتين أو مزدوجتين</td></tr>
<tr><td>(10|11|12|0?[1–9])(?<sep>[-/])(30|31|2\d|1\d|0?[1–9])\k<sep>(\d{4}|\d{2})</td><td>تاريخ بالتنسيق الأمريكي

mm-dd-yyyy OR mm/dd/yyyy</td></tr>
<tr><td>(30|31|2\d|1\d|0?[1–9])(?<sep>[-/])(10|11|12|0?[1–9])\<sep>(\d{4}|\d{2})</td><td>تاريخ بالتنسيق الأوربي

dd-mm-yyyy or dd/mm/yyyy</td></tr>
<tr><td>(2[0–3]|[01]\d|\d):[0–5]\d</td><td>تاريخ بنظام 24 ساعة والصفر السابق لرقم الساعة اختياري

hh:mm 24-hour format</td></tr>
<tr><td>\(\d{3}\)–\d{3}–\d{4}</td><td>رقم هاتف بالشكل

(123)-456-7890.</td></tr>
<tr><td>\d{5}(-\d{4})?</td><td>رمز منطقة أمريكي</td></tr>
<tr><td>\d{3}-\d{2}-\d{4}</td><td>رقم ضمان اجتماعي أمريكي</td></tr>
<tr><td>((\d{16}|\d{4}(-\d{4}){3})|(\d{4}(\d{4}){3}))</td><td>رقم بطاقة ائتمان من 16 خانة</td></tr>
<tr><td>([0–9A–Fa–f]{32}|[0–9A–Fa–f]{8}-([0–9A–Fa–f]{4}-){3}[0–9A–Fa–f]{12})</td><td>GUID</td></tr>
<tr><td>([A–Za–z]:)?\\?([^/:*?<>"|\\]+\\)*[^/:*?<>"|\\]+</td><td>اسم ملف مع أو بدون المسار</td></tr>
<tr><td>(http|https)://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?</td><td>An Internet URL</td></tr>
<tr><td>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*</td><td>An Internet e-mail address.</td></tr>
</table>

<table>
<tr><th>Pattern</th><th>الوصف</th></tr>
<tr><td>((25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)\.){3}(25[0-5]|2[0-4]\d|1\d\d|[1-9]\d|\d)</td><td>A four-part IP address, such as 192.168.0.1; the pattern verifies that each number is in the range 0–255.</td></tr>
<tr><td>([1-5]\d{4}|6[0-4]\d{3}|65[0-4]\d{2}|655[0-2]\d|6553[0-4]|\d{1,4})</td><td>A 16-bit integer that can be assigned to a UShort variable, in the range of 0 to 65,535.</td></tr>
<tr><td>(-?[12]\d{4}|-?3[0-1]\d{3}|-?32[0-6]\d{2}|-?327[0-5]\d|-?3276[0-7]|-32768|-?\d{1,4})</td><td>A 16-bit integer that can be assigned to a Short variable, in the range of –32,768 to 32,767.</td></tr>
<tr><td>^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])\w{8,}$</td><td>كلمة سر تحتوي ثمانية محارف على الأقل بحيث تحتوي على الأقل رقما وحرا كبيرا وحرفا صغيرا ويمكن أن يحتوي بعض الحروف الخاصة</td></tr>
</table>

القسم الثالث عشر ـ مواضيع مختلفة

ويضم المواضيع التالية:

- If Operator
- Lambda Expressions
- تعابير لمدا في العمق Lambda Expressions
- Nullable Value Types
- Object Initializers
- الاستدلال المحلي على النوع Local Type Inference
- إجبار المستخدم على اختيار واحدة من عدة قيم محددة سابقا في صندوق النصوص
- استخدام الوظائف المخزنة Using Stored Procedures
- الأنواع المجهولة Anonymous Types
- التحكم PropertyGrid
- التحويل بين أنواع البيانات باستخدام التضييق Explicit والتوسيع Implicit
- الفئة StringBuilder
- الواصفة Obsolete Attribute
- تخزين ملف ما ضمن Exe البرنامج أثناء التطوير واستعادته أثناء التشغيل
- تشفير الأسرار للمستخدم الحالي
- توجيهات المعالج
- كيف تجعل لنافذة برنامج ظلا
- كيف تقوم بعمل أيقونة خاصة لتحكمك الخاص
- لماذا يأخذ كودك وقتا طويلا أثناء التنفيذ
- ملفات المصادر وتخصيص البرنامج محليا Resources and localization

If Operator

في البداية أحب أن أنوه أن If Operator هنا مختلف عن الوظيفة IIf أو عبارة If … Then … Else المعتادة. فسابقا عندما كنا نستخدم IIF كان النوع المعاد من النوع Object مما يعني أنه لن يحصل تدقيق على النوع بالحالة الافتراضية كما لن يتوفر IntelliSense لتلك القيمة ولهؤلاء الذين يصرون على كتابة كود آمن من خلال الأنواع ومن أجل الربط المبكر للكود كان يجب عليهم تحويل ذلك النوع إلى نوع البيانات المراد فقد يبدو لديهم الكود كما يلي

```
Dim intC As Integer = CInt(IIf(intA = intB, intA, intB - 1))
```

ولكن الآن وباستخدام المعامل If يمكننا إعادة كتابة نفس الكود والحصول على فوائد الربط المبكر وتدقيق النوع وتوفر intelliSense لتلك القيمة

```
Dim intD As Integer = If(intA = intB, intA, intB)
```

الصيغة العامة

يوفر المعامل If طريقة مختصرة لإعادة قيمة واحدة من قيمتين وفق شرط معين حيث يمكن استدعاؤه بتمرير ثلاثة وسائط له أو وسيطتين وتكون الصيغة العامة له

```
If( [argument1,] argument2, argument3 )
```

استدعاء If Operator بثلاثة وسائط

عندما يتم استدعاء المعامل If بثلاثة وسائط يجب أن يمكن تقييم الوسيطة الأولى كقيمة بوليانية التي ستحدد بدورها أية واحدة من الوسيطتين الأخريين سيتم تقييمها وإعادة قيمتها وتكون الوسائط الثلاث عند استخدام المعامل If بثلاثة وسائط كما يلي:

- argumet1 ضرورية وقيمتها بوليانية وهي تحدد أية واحدة من الوسيطتين الأخريين سيتم تقييمها وإعادتها
- argumet2 ضرورية وهي من النوع Object ويتم تقييمها وإعادتها في حال كون قيمة argument1 هي True
- argument3 ضرورية وهي من النوع Object ويتم تقييمها وإعادتها في حال كون قيمة argument1 هي False

فعندما يتم استدعاء المعامل If بثلاثة وسائط يعمل بشكل مشابه للوظيفة IIf فيما عدا أنها تستخدم التقييم المختصر. فالوظيفة IIf تقيم دوما جميع الوسائط الثلاث بينما المعامل If يقوم بتقييم اثنين فقط من تلك الوسائط حيث يتم تقييم الوسيطة الأولى وتحويل نوعها إلى Boolean فإن كانت النتيجة True فسيتم تقييم argument2 ثم إعادة قيمتها ولكن لن يتم تقييم argument3 في هذه الحالة. وإن كانت قيمة الوسيطة الأولى False عندها لن يتم تقييم argument2 وسيتم تقييم argument3 وإعادة قيمتها ويوضح المثال التالي استخدام المعامل If بثلاثة وسائط

```
' This statement prints TruePart, because the first argument is true.
Console.WriteLine(If(True, "TruePart", "FalsePart"))

' This statement prints FalsePart, because the first argument is false.
Console.WriteLine(If(False, "TruePart", "FalsePart"))

Dim number = 3
' With number set to 3, this statement prints Positive.
Console.WriteLine(If(number >= 0, "Positive", "Negative"))

number = -1
' With number set to -1, this statement prints Negative.
Console.WriteLine(If(number >= 0, "Positive", "Negative"))
```

والمثال التالي يوضح قيمة التقييم المختصر مظهرا محاولتين لتقسيم متغير number على متغير divisor عدا أنه عندما تكون قيمة divisor مساوية للصفر يجب إعادة القيمة صفر و لا يجب محاولة القيام بعملية القسمة وإلا نتج عن ذلك خطأ وقت التنفيذ – خطأ القسمة على صفر - وبسبب أن المعامل If يستخدم التقييم المختصر فإنه يقيم إما الوسيطة الثانية أو الثالثة اعتمادا على قيمة الوسيطة الأولى فإن كانت للوسيطة الأولى القيمة True فيكون divisor لا يحمل القيمة صفر وبالتالي يكون آمنا تقييم الوسيطة الثانية وإجراء عملية القسمة وإن كانت قيمة الوسيطة الأولى False فسيتم تقييم الوسيطة الثالثة فقط وسيتم إعادة القيمة صفر ولهذا فعندما تكون قيمة divisor مساوية للصفر فلن يتم محاولة إجراء عملية القسمة وبالتالي لن يكون هناك خطأ في زمن التنفيذ وبما أن IIf لا تستخدم التقييم المختصر فسيتم تقييم الوسيطة الثانية دوما مهما كانت قيمة الوسيطة الأولى وبالتالي سينطلق خطأ القسمة على صفر وقت التنفيذ دوما

```
number = 12

' When the divisor is not 0, both If and IIf return 4.
Dim divisor = 3
Console.WriteLine(If(divisor <> 0, number \ divisor, 0))
Console.WriteLine(IIf(divisor <> 0, number \ divisor, 0))

' When the divisor is 0, IIf causes a runtime error, but If does not.
divisor = 0
Console.WriteLine(If(divisor <> 0, number \ divisor, 0))
' Console.WriteLine(IIf(divisor <> 0, number \ divisor, 0))
```

استدعاء If Operator بوسيطتين

يمكن حذف الوسيطة الأولى مما يمكنك من استدعاء المعامل If بوسيطتين حيث تكونان كما يلي

- argument2 ضرورية من النوع Object ويجب أن تكون من نوع يمكن أن يحمل القيمة Nothing أو نوع مرجعي Reference or nullable type حيث يتم تقييمه وإعادته عندما يحمل أي قيمة مغايرة لـ Nothing
- argument3 ضرورية من النوع Object حيث يتم تقييمها وإعادة قيمتها في حالة كون قيمة argument2 مساوية لـ Nothing

فعندما يتم حذف الوسيطة البوليانية عندها يجب أن تكون الوسيطة الأولى من نوع يقبل أن يحمل القيمة Nothing أو نوع مرجعي reference or nullable type فإن تم تقييم الوسيطة الأولى إلى Nothing عندها يتم إعادة قيمة الوسيطة الثانية وفي جميع الحالات الأخرى يتم إعادة قيمة الوسيطة الأولى كما يظهر المثال التالي

```
' Variable first is a nullable type.
Dim first? As Integer = 3
Dim second As Integer = 6

' Variable first <> Nothing, so its value, 3, is returned.
Console.WriteLine(If(first, second))

second = Nothing
' Variable first <> Nothing, so the value of first is returned again.
Console.WriteLine(If(first, second))

first = Nothing
second = 6
' Variable first = Nothing, so 6 is returned.
Console.WriteLine(If(first, second))
```

Lambda Expressions

الـ Lambda Expression هو وظيفة Function بدون اسم تحتسب وتعيد قيمة وحيدة كما يمكن استخدامها في التعابير التي تطلب إجراءات مفوضة Delegate والمثال التالي عن هذه التعابير يأخذ قيمة ويعيد الناتج بعد إضافة واحد لها

```
Function (num As Integer) num + 1
```

كما يمكنك إسناد هذه الوظيفة لمتغير وتمرير القيمة له

```
Dim add1 = Function(num As Integer) num + 1
Console.WriteLine(add1(5))
```

كما يمكنك تعريف وتنفيذ الوظيفة بنفس الوقت

```
Console.WriteLine((Function(num As Integer) num + 1)(5))
```

كما يمكن أن تستخدم Lambda Expressions كقيمة معادة عند استدعاء وظيفة أو تمريرها كوسيط لإجراء مفوض ففي المثال التالي تستخدم Lambda Expressions بوليانية كوسائط للإجراء testResult حيث تطبق الطريقة فحص بولياني لوسيط من النوع Integer ويظهر القيمة Success إذا كانت قيمة Lambda Expression هي True أو Failure إن كانت قيمته False

```
Module Module2

    Sub Main()
        ' The following line will print Success, because 4 is even.
        testResult(4, Function(num) num Mod 2 = 0)
        ' The following line will print Failure, because 5 is not > 10.
        testResult(5, Function(num) num > 10)
    End Sub

    ' Sub testResult takes two arguments, an integer value and a
    ' Boolean function.
    ' If the function returns True for the integer argument, Success
    ' is displayed.
    ' If the function returns False for the integer argument, Failure
    ' is displayed.
    Sub testResult(ByVal value As Integer, ByVal fun As Func(Of Integer, _
                         Boolean))

        If fun(value) Then
            Console.WriteLine("Success")
        Else
            Console.WriteLine("Failure")
        End If
    End Sub

End Module
```

تكون تعابير Lambda Expressions هي الأساس لكثير من معاملات الاستعلام Linq حيث يقوم المترجم Compiler بإنشاء تعابير Lambda Expressions للقيام بالعمليات الحسابية للطرائق الخاصة بالاستعلام مثل Where و Select و Order By و Take While فعلى سبيل المثال انظر الاستعلام التالي

```
Dim londonCusts = From cust In db.Customers
                  Where cust.City = "London"
                  Select cust
```

حيث ستتم ترجمته إلى الكود التالي

```
Dim londonCusts = db.Customers _
    .Where(Function(cust) cust.City = "London") _
    .Select(Function(cust) cust)
```

وتكون صيغتها على الشكل

- هذه التعابير لا تملك اسما
- لا يمكن استخدام المعدلات معها مثل Overloads أو Overrides
- لا تستخدم قسم AS لتحديد نوع القيمة المعادة وبدلا عن ذلك يكون نوع القيمة المعادة هو نوع القيمة التي يشكلها جسم الإجراء فإن كان جسم الإجراء مثلا "Cust.City = "London فتكون القيمة المعادة بوليانية
- جسم الإجراء يجب أن يكون تعبير وليس تصريح ويمكن أن يحتوي على استدعاء لوظيفة Function ولكنه لا يمكن أن يستدعي إجراء Sub
- لا يوجد تعبير Return وتكون القيمة المعادة هي قيمة ذلك التعبير الذي يشكل جسم الوظيفة
- لا يوجد تعبير End
- يجب أن تكون جميع الوسائط محددة النوع أو تكون جميعها بأنواع بالإشارة
- غير مسموح بالوسائط الاختيارية
- الوسائط Generic غير مسموح بها

ونتيجة لهذه القواعد سنرى أن أي تعبير Lambda Expression سيكون بسيطا وغير معقد

تشترك Lambda Expression مع الوظائف Methods بأنها محددة ولها جميع حقوق الوصول كأي كود مكتوب في الطريقة التي تحتويها وهذا يتضمن الوصول إلى متغيرات الأعضاء والوظائف وجميع المغيرات الموجودة في الوظيفة التي تحتوي التعبير Lambda Expression ففي المثال التالي المتغير target هو محلي بالنسبة لـ makeTheGame والطريقة التي تم تحديد التعبير Lambda Expression فيها هي playTheGame لاحظ أن القيمة المعادة من التعبير Lambda Expression يتم تعيينها لـ takeAGuess في Main مازالت تستطيع الوصول للمتغير المحلي target

```
Module Module1

    Sub Main()
        ' Variable takeAGuess is a Boolean function. It stores the target
        ' number that is set in makeTheGame.
        Dim takeAGuess As gameDelegate = makeTheGame()

        ' Set up the loop to play the game.
        Dim guess As Integer
        Dim gameOver = False
        While Not gameOver
          guess = CInt(InputBox("Enter a number between 1 and 10 (0 to quit)", _
                                "Guessing Game", "0"))
            ' A guess of 0 means you want to give up.
            If guess = 0 Then
                gameOver = True
            Else
   ' Tests your guess and announces whether you are correct. Method takeAGuess
   ' is called multiple times with different guesses. The target value is not
   ' accessible from Main and is not passed in.
                gameOver = takeAGuess(guess)
                Console.WriteLine("Guess of " & guess & " is " & gameOver)
            End If
        End While

    End Sub

    Delegate Function gameDelegate(ByVal aGuess As Integer) As Boolean

    Public Function makeTheGame() As gameDelegate
```

```
        ' Generate the target number, between 1 and 10. Notice that
        ' target is a local variable. After you return from makeTheGame,
        ' it is not directly accessible.
        Randomize()
        Dim target As Integer = CInt(Int(10 * Rnd() + 1))

        ' Print the answer if you want to be sure the game is not cheating
        ' by changing the target at each guess.
        Console.WriteLine("(Peeking at the answer) The target is " & target)

       ' The game is returned as a lambda expression. The lambda expression
       ' carries with it the environment in which it was created. This
       ' environment includes the target number. Note that only the current
       ' guess is a parameter to the returned lambda expression, not the target.

        ' Does the guess equal the target?
        Dim playTheGame = Function(guess As Integer) guess = target

        Return playTheGame

    End Function

End Module
```

ويستعرض المثال التالي مجالا عريضا من حقوق الوصول المعششة في Lambda Expression فعندما يتم تنفيذ التعبير Lambda Expression من Main كـ aDel يستخدم العناصر التالية (حقل في الفئة aField – خاصية في الفئة aProp – وسيط للإجرائية functionWithNestedLambda هو level1 – متغير محلي لـ functionWithNestedLambda هو localVar – وسيط للتعبير Lambda Expression المعشش هو level2)

```
Module Module3

    Sub Main()
        ' Create an instance of the class, with 1 as the value of
        ' the property.
        Dim lambdaScopeDemoInstance = New LambdaScopeDemoClass _
            With {.Prop = 1}

        ' Variable aDel will be bound to the nested lambda expression
        ' returned by the call to functionWithNestedLambda.
        ' The value 2 is sent in for parameter level1.
        Dim aDel As aDelegate = _
            lambdaScopeDemoInstance.functionWithNestedLambda(2)

        ' Now the returned lambda expression is called, with 4 as the
        ' value of parameter level3.
        Console.WriteLine("First value returned by aDel:   " & aDel(4))

        ' Change a few values to verify that the lambda expression has
        ' access to the variables, not just their original values.
        lambdaScopeDemoInstance.aField = 20
        lambdaScopeDemoInstance.Prop = 30
        Console.WriteLine("Second value returned by aDel: " & aDel(40))
    End Sub

    Delegate Function aDelegate(ByVal delParameter As Integer) _
        As Integer

    Public Class LambdaScopeDemoClass
        Public aField As Integer = 6
        Dim aProp As Integer
```

```
        Property Prop() As Integer
            Get
                Return aProp
            End Get
            Set(ByVal value As Integer)
                aProp = value
            End Set
        End Property

        Public Function functionWithNestedLambda _
           (ByVal level1 As Integer) As aDelegate
            Dim localVar As Integer = 5

            ' When the nested lambda expression is executed the first
            ' time, as aDel from Main, the variables have these values:
            ' level1 = 2
            ' level2 = 3, after aLambda is called in the Return statement
            ' level3 = 4, after aDel is called in Main
            ' locarVar = 5
            ' aField = 6
            ' aProp = 1
            ' The second time it is executed, two values have changed:
            ' aField = 20
            ' aProp = 30
            ' level3 = 40
            Dim aLambda = Function(level2 As Integer) _
                              Function(level3 As Integer) _
                                  level1 + level2 + level3 + localVar _
                                  + aField + aProp

            ' The function returns the nested lambda, with 3 as the
            ' value of parameter level2.
            Return aLambda(3)
        End Function

    End Class
End Module
```

كما يمكن تحويل Lambda Expressions لتتوافق مع الإجراءات المفوضة فعندما تعين Lambda Expression لإجراء مفوض Delegate يمكنك تحديد أسماء الوسائط ولكن مع إغفال أنواع البيانات الخاصة بها تاركا مهمة تحديدها للإجراء المفوض ففي المثال التالي يتم تعيين Lambda Expression لمتغير اسمه del من النوع ExampleDel الذي هو عبارة عن إجراء مفوض يأخذ وسيطتين integer و string لاحظ أن أنواع المتغيرات في Lambda Expression لم يتم تحديدها ومع ذلك فـ del يتطلب وسيطا من النوع integer ووسيطا آخر من النوع string كما تم تحديده عند تعريف ExampleDel

```
' Definition of function delegate ExampleDel.
Delegate Function ExampleDel(ByVal arg1 As Integer, _
                             ByVal arg2 As String) As Integer

' Declaration of del as an instance of ExampleDel, with no data
' type specified for the parameters, m and s.
Dim del As ExampleDel = Function(m, s) m

' Valid call to del, sending in an integer and a string.
Console.WriteLine(del(7, "up"))

' Neither of these calls is valid. Function del requires an integer
' argument and a string argument.
'Console.WriteLine(del(7, 3))
'Console.WriteLine(del("abc"))
```

في المثال التالي يتم تحديد Lambda Expression ليعيد القيمة True إذا كان الوسيط يمتلك قسمة أو False إذا كان القيمة Nothing

```
Dim notNothing = Function(num? As Integer) _
             num IsNot Nothing
Dim arg As Integer = 14
Console.WriteLine("Does the argument have an assigned value?")
Console.WriteLine(notNothing(arg))
```

والمثال التالي يحدد Lambda Expression يعيد Index العنصر الأخير في مصفوفة

```
Dim numbers() As Integer = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim lastIndex = Function(intArray() As Integer) _
                    intArray.Length - 1
For i = 0 To lastIndex(numbers)
    numbers(i) = numbers(i) + 1
Next
```

تعابير لمدا في العمق Lambda Expressions

تعتبر تعابير لمدا من الإضافات المفيدة في فيجول بايزيك 2008 حيث يمكنك إعادتها كقيمة من وظيفة أو تمريرها كمحدد لوظيفة أخرى حيث تمت إضافتها للغة البايزيك كدعم للغة الاستعلامات المضمنة Linq التي تضيف إمكانيات استعلامية قوية لبرمجة البيانات في فيجول بايزيك وعندما تبدأ باستخدام تعابير لمدا سترى القوة والمرونة الكامنة فيها

ما هي تعابير لمدا

يشكل الكود التالي مثالا عن تعريف تعبير لمدا أساسي فهو يعرف تعبير لمدا يأخذ Integer ويعيد Integer بحيث يأخذ قيمة الدخل ويعيدها مضروبة بـ 2

```
Dim doubleIt as Func(Of Integer, integer) = _
     Function(x As Integer) x * 2
```

والنوع Func من الأنواع الجديدة في فيجول بايزيك 2008 وهو في الأساس إجراء مفوض Delegate يعيد نوعا يحدده المحدد الأخير ويمكنك من تمرير أربعة محددات تسبق ذلك المحدد والنوع المفوض Func معرف في المجمع System.Core.dll الأمر الذي يمكنك من الاستفادة منه فوريا وذلك لأن المجمع المذكور يتم استيراده تلقائيا عندما ننشئ تطبيقا جديدا ويمثل الكود التالي تحميلات Overloads مختلفة لـ Func

```
Dim f0 As Func(Of Boolean)
Dim f1 As Func(Of Integer, Boolean)
Dim f4 As Func(Of Integer, Integer, Integer, Integer, Boolean)
```

ففي المثال السابق f0 هو مفوض يعيد قيمة Boolean و f1 مفوض يأخذ Integer ويعيد Boolean و f4 مفوض يأخذ أربعة محددات من النوع Integer ويعيد قيمة من النوع Boolean وتكمن النقطة الأساسية في التعبير لمدا هو أنه مفوض قابل للاستدعاء تماما كالمفوضات في فيجول بايزيك 2005 فمن ناحية المساواة اليمنى في قطعة الكود الأولى يمكنك رؤية الصيغة الجديدة للتعابير لمدا فهي تبدأ بالكلمة المحجوزة Function متبوعة بقائمة من المحددات وتعبير وحيد ففي المثال السابق يأخذ تعبير لمدا محددا واحدا من النوع Integer ونلاحظ عدم وجود تعبير Return وذلك لأن المترجم يعرف النوع المؤسس عليه التعبير وبهذا يقوم بتمرير عبارة Return تلقائيا وفي هذه الحالة بما أن x هو من النوع Integer ونتيجة المعادلة هي Integer لهذا فنتيجة تعبير لمدا هي Integer أيضا ويمكن السحر في تعابير لمدا في أنه يمكن استخدامها كمفوض بسيط كما نرى في المثال

```
Dim doubleIt As Func(Of Integer, Integer) = _
     Function(x As Integer) x * 2
Dim z = doubleIt(20)
```

فإن نفذت الكود السابق سترى أن القيمة المخزنة في z هي 40 وأنت بهذا قمت بإنشاء تعبير لمدا يضاعف قيمة أي Integer يمرر له.

دعنا الآن نتفحص مثال معقد أكثر باستخدام تعابير لمدا

```
Dim mult As Func(Of Integer, Func(Of Integer, Integer)) = _
     Function(x As Integer) Function(y As Integer) x * y
```

ويعتبر mult تعبير لمدا معقد قليلا فهو يأخذ كدخل له محدد من النوع Integer ويعيد تعبير لمدا كقيمة له والذي أخذ بدوره قيمة Integer ويعيد قيمة Integer كما يمكننا إعادة تقسيم التعبير السابق على أسطر من أجل توضيح الكود

```
Dim mult As Func(Of Integer, Func(Of Integer, Integer)) = _
     Function(x As Integer) _
          Function(y As Integer) x * y
```

فتعبير لمدا الخارجي يحتوي تعبير لمدا آخر الذي يستخدم من قبل المترجم كقيمة معادة ويكون التوقيع الخاص بتعبير لمدا الداخلي مماثلا لتوقيع المفوض Func(Of Integer, Integer) في القيمة المعادة من تعبير لمدا الخارجي حيث يقوم المترجم بترجمة التعبير بأكمله دون مشاكل ويمكننا رؤية تعبير لمدا هذا كما يلي

```
Dim mult_10 = mult(10)
Dim r = mult_10(4)
```

فالسطر الأول يحدد mult_10 كـ mult(10) وبما أن Mult(10) يعيد تعبير لمدا يأخذ محدد ويضربه بـ 10 والنوع المعاد من mult_10 هو Func(Of Integer, Integer) والسطر الثاني يستدعي mult_10 ممررا له القيمة 4 بهذا ستكون القيمة المخزنة في r هي 40 ويكون نوع r هو Integer ويعتبر mult مصنع لتعابير لمدا فهو يعيد تعبير لمدا مخصص بالمحدد الأول وستلاحظ أن تعبير لمدا الداخلي يستخدم محدد تعبير لمدا الخارجي ولكن فترة حياة تعبير لمدا الداخلي تتجاوز فترة حياة تعبير لمدا الخارجي

تعابير لمدا كاستدعاءات

بما أن تعابير لمدا هي ببساطة مفوضات لذا يمكنك استخدامها في أي مكان يمكن استخدام المفوض فيه. لاحظ الإجراء التالي الذي يأخذ مفوض كمحدد له ويستدعي مفوض من أجل كل عنصر في القائمة

```
Delegate Function ShouldProcess(Of T) (element As T) As Boolean

Sub ProcessList(Of T) ( _
            Elements As List(Of T), shouldProcess As ShouldProcess(Of T))

    For Each elem in elements
        If shouldProcess(elem) Then
            ' Do some processing here
        End If
    Next
End Sub
```

ويكون المثال التالي تطبيقا قياسيا على المفوضات فالطريقة ProcessList ستمر على كل عنصر من القائمة وتتحقق فيما إذا كان عليها معالجة العنصر ثم تقوم ببعض المعالجة وحتى تتمكن من استخدام هذا في فيجول بايزيك 2005 عليك إنشاء وظيفة تمتلك نفس توقيع المفوض ثم تمرر عنوان تلك الوظيفة إلى الإجراء ProcessList

```
Class Person
    Public age As Integer
End Class

Function _PrivateShouldProcess(person As Person) As Boolean
    Return person.age > 50
End Function

Sub DoIt()
    Dim list As New List(Of Person)
    'Obtain list of Person from a database, for example
    ProcessList(list, AddressOf _PrivateShouldProcess)
End Sub
```

وهذا يسبب بعض الإزعاج فغالبا عليك البحث في توثيق الكود لمعرفة ماذا يمثل توقيع المفوض ثم يجب عليك مطابقته كليا وإن احتجت لاستدعاء ProcessList مع عدة إجراءات ستقوم بإنشاء العديد من الوظائف الخاصة.

دعنا نرى الآن كيف يمكننا استدعاء هذا الإجراء باستخدام تعابير لمدا

```
Class Person
    Public age As Integer
End Class
```

```
Sub DoIt()
    Dim list As new List(Of Person)
    'Obtain list of Person from a database, for example
    ProcessList(list, Function(person As Person) person.age > 50)
End Sub
```

فباستخدام تعابير لمدا لم يعد هناك حاجة لإنشاء وظيفة خاصة للقيام بمنطق المعالجة حيث يتم تعريف المفوض في النقطة التي سيستخدم فيها وهذا أفضل من تعريفه ضمن وظيفة خاصة في مكان ما وفقدان محليتها باستخدام الطريقة الخاصة وبهذا أنت ترى قوة تعابير لمدا وتسهيلها لعملية قراءة وصيانة الكود الخاص بك

لماذا تم تقديم تعابير لمدا

من أجل دعم استعلامات لينك Linq كان يجب إضافة مجموعة من الإمكانيات الجديدة للغة فيجول بايزيك ومن ضمنها كانت تعابير لمدا. افترض أنه لدينا الاستعلام التالي

```
Dim q = From p In Process.GetProgesses() _
        Where p.PriorityClass = ProcessPriorityClass.High _
        Select P
```

فلكي يتم ترجمة هذا التعبير يجري الكثير من العمل تحت الغطاء فالمترجم سيقوم بالمرور عبر المجموعة Process.GetProcesses ويطبق المرشح الموجود في قسم Where عليها ويعيد قائمة بالعمليات التي تطابق ذلك الشرط كما نلاحظ وجود تعبير فيجول بايزيك داخل قسم Where هو p.PriorityClass = ProcessPriorityClass.High وذلك لتطبيق المرشح وهنا يقوم المترجم بإنشاء تعبير لمدا من أجل المرشح الموجود في قسم Where ويطبقه على كل عنصر في قائمة العمليات

```
Dim q = Process.GetProcesses().Where( _
    Function(p) p.PriorityClass = ProcessPriorityClass.High)
```

وأساسا يشكل التعبير لمدا اختصارا للمترجم من أجل اختصار عملية إنشاء الطرق وربطها مع المفوضات حيث يقوم بكل ذلك من أجلك والفائدة التي نجنيها من تعابير لمدا ولا نجنيها عند استخدام الوظائف والمفوضات هي أن المترجم هنا يستخدم الاستدلال المحلي على النوع على تعابير لمدا ففي المثال السابق يتم تحديد نوع المحدد p بناء على الاستخدام وفي هذه الحالة يحدد التعبير في قسم Where تعبير لمدا ويقوم المترجم بالاستدلال آليا على نوع القيمة المعادة من التعبير لمدا بحيث تعتبر ميزة الاستدلال المحلي على النوع المدعومة من قبل المعالج من الإضافات القوية لفيجول بايزيك

الاستدلال المحلي على النوع

تقديم ميزة الاستدلال المحلي على النوع القوية يعني أنه لم يعد عليك أن تقلق حول تحديد النوع الملائم لكل متغير وبالتالي فهي تمكنك من القيام بالعديد من الأمور التي كانت تبدو مستحيلة فالاستدلال على النوع المعاد من تعابير لمدا مفيد جدا فإن كان لديك نوع مفوض تريد ربطه مع تعبير لمدا لم يعد عليك تحديد نوع جميع المحددات

```
Dim lambda As Func(Of Integer, Integer) = Function(x) x * x
```

ففي هذا المثال يكون نوع تعبير لمدا هو Func(Of Integer, Integer) وهو مفوض يأخذ محدد من النوع Integer ويعيد محدد من النوع Integer وكنتيجة لهذا فالمترجم يستدل آليا على أن المحدد x العائد لتعبير لمدا هو من النوع Integer والقيمة المعادة من التعبير لمدا هي Integer أيضا كما يمكنك الاستفادة من الاستدلال على نوع تعابير لمدا عندما تستدعي طريقة تأخذ مفوضا لاحظ الكود التالي

```
Delegate Function ShouldProcess(Of T) (element As T) As Boolean

Sun ProcessList(Of T) (_
            Elements As List(Of T), shouldProcess As ShouldProcess(Of T))
    ' Method body removed for brevity
End Sub
```

في هذه الحالة تأخذ الوظيفة ProcessList تعبير لمدا ويمكن استدعاؤها على الشكل

```
Sub DoIt()
    Dim list As new List(Of A)
    ' fill or obtain elements in list
    ProcessList(list, Function(a) a.x > 50)
End Sub
```

لاحظ أننا لم نحدد نوع المحدد الممرر للتعبير لمدا كما فعلنا سابقا وذلك لأن المعالج يستدل عليه بنفسه.

كيف يمكن حدوث شئ كهذا؟ في الحقيقة هناك عدة مستويات من الاستدلال على النوع في هذا المثال ففي البداية يرى المترجم ProcessList كإجراء عادي يأخذ list(Of T) كدخل له و ShouldProcess(Of T) في استدعاء ProcessList ويرى المترجم أن list هي المحدد الأول وأنها list(Of Person) وبما أن المحدد الثاني لا يوفر تلميحات حول ماهية نوع T فيقرر المترجم أن T من النوع Person ويستدل من هذا على أن محدد ShouldProcess(Of T) هو من النوع Person وبهذا يستدل على أن المحدد الثاني هو من النوع ShouldProcess(Of T) وأخيرا بما أن تعبير لمدا لا يقدم نوع المحدد الخاص به والمترجم يعرف أن نوع المحدد يعتمد على توقيع المفوض ShouldProcess(Of T) وقد استدل على أن نوع المحدد a هو Person ويعتبر هذا نوعا قويا من الاستدلال على النوع فليس عليك معرفة نوع محددات المفوض عندما تبني تعبير لمدا وفي الحقيقة من الأفضل ترك المترجم يقوم بذلك العمل نيابة عنك والاستدلال على نوع النتيجة بهذه الطريقة مفيد حقيقة إن لم يكن لديك نوع مفوض وتريد من المترجم أن يقوم بتصنيعه من أجلك علما بأن هذه الميزة متوفرة في فيجول بايزيك فقط

```
Dim lambda = Function(x As Integer) x * x
```

ففي المثال السابق بما أن المحدد x هو من النوع Integer فالمترجم يستدل آليا على أن القيمة المعادة هي من النوع Integer أيضا كنتيجة المعادلة الموجودة في التعبير وبما أن تعبير لمدا لا يمتلك نوعا لهذا يقوم المترجم بتصنيع مفوض مجهول يطابق شكل تعبير لمدا ويربط ذلك النوع المفوض بتعبير لمدا. وهذه ميزة عظيمة لأنها تعني أنه يمكنك إنشاء تعابير لمدا بسرعة بدون أن تحتاج لتعريف الأنواع المفوضة الخاصة بها. فكم مرة كنت في وضع تحتاج فيه لتطبيق مجموعة من المتغيرات وتحتاج إلى فعل ذلك في العديد من الأماكن ففي الكود التالي مرت عدة حالات مشابهة وعادة يمكننا معالجة ذلك بحيث يمكن التحقق من الشرط في مكان واحد بدلا من التشتت في أرجاء الوظيفة

```
Class Motorcycle
    Public color As String
    Public CC As Integer
    Public weight As Integer
End Class

Sub PrintReport(motorcycle As New Motorcycle)
    If motorcycle.color = "Red" And motorcycle.CC = 600 And _
        Motorcycle.weight > 300 And Motorcycle.weight < 400 Then

        ' do something here
    End If

    ' do something here

    If motorcycle.color = "Red" And motorcycle.CC = 600 And _
        Motorcycle.weight > 300 And Motorcycle.weight < 400 Then

        ' do something here
    End If
End Sub
```

وفي بعض الأحيان يستخدم هذا التحقق في هذه الوظيفة فقط ويمكننا إضافة إجراء في الفئة لدعم تلك الوظيفة فقط والقيام بذلك يؤثر على عملية صيانة الكود فماذا لو قام أحد ما باستدعاء هذه الوظيفة في مكان آخر واحتجت للقيام بتعديل ما وقد يؤدي هذا في بعض الفئات إلى وجود وظائف يصعب تعقبها جاعلا خاصية IntelliSense أقل فائدة لوجود العديد من المدخلات الإضافية فيها إضافة إلى خرق منطق

المحلية وإن قمنا بذلك باستخدام طريقة منفصلة مختلفة عندها يفضل أن تكون قريبة من الطريقة التي تستخدمها ومع وجود العديد من الأشخاص يعملون على نفس المشروع يصبح من الصعب صيانة المحلية على المدى الطويل وهنا يأتي استخدام تعابير لمدا وترك المترجم يقوم آليا بإنشاء المفوضات ويقوم باستخدامها عند الحاجة

```
Sub PrintReport(motorcycle As New Motorcycle)
    Dim check = Function(m As Motorcycle) m.color =  "Red" And _
                                          m.CC = 600 And _
                                          m.weight > 300 And _
                                          m.weight < 400

    If check(motorcycle) Then
        ' do something here
    End If

    ' do something here

    If check(motorcycle) Then
        ' do something here
    End If
End Sub
```

قمنا هنا بتعديل منطق تفحص بعض شروط Motorcycle ليستخدم تعابير لمدا عوضا عن سيئات الطرائق الخاصة حيث سيقوم المترجم تلقائيا بإنشاء النوع المفوض ويقوم بالعمل لكي نستطيع استدعاء تعابير لمدا أينما احتاج ذلك وهذه الطريقة مفيدة لأنها تضع المنطق قريب من التصريح حيث نقوم بتصنيع نسخة واحدة ويقوم المترجم بعدها بمعظم عمليات الصيانة ويعتبر هذا مفيدا لأنه يمكنك من بناء تعبير معقد كجسم لتعبير لمدا وباستخدام الربط المتأخر والاستدلال على النوع في هذا السيناريو فلا نحدد نوع تعبير لمدا أو المتغير

```
Dim lambda = Function(x) x * x
```

وهنا أيضا يولد المعالج مفوض مجهول من أجلك ولكن يحدد نوع تعبير لمدا كـ System.Object وهذا يعني أنه قد تم تفعيل الربط المتأخر في هذا السيناريو عندما يكون الخيار Option Strict على الوضع Off ويعتبر هذا السيناريو جيدا بالنسبة لأولئك الذين يعتمدون على الربط المتأخر حيث أن تعابير لمدا تدعم عمليات الربط المتأخر بشكل كامل ففي المثال السابق طالما أن المعامل * معرف على الأنواع الممررة إلى تعبير لمدا فسوف يعمل

```
Dim a = lambda(10)
Dim b = lambda(CDec(10))
Dim c = lambda("This will throw an exception because " & _
               "strings don't support the * operator")
```

وكما ترى من المثال السابق طالما أن المعامل * موجود في مكتبات زمن التشغيل بالنسبة للنوع الممرر فسوف يجري كل شئ بشكل جيد كما أن تعابير لمدا تتأقلم بشكل رائع مع الربط المتأخر في فيجول بايزيك.

توليد الكود تحت الغطاء

بعدما استكشفنا تعابير لمدا دعنا نلقي نظرة على الكود الذي يتم توليده من قبل المترجم. انظر للكود السابق

```
Sub TestLambda()
    Dim doubleIt As Func(Of Integer, Integer) = _
        Function(x As Integer) X * 2
    Console.WriteLine(doubleIt(10))
End Sub
```

أنت تعلم أن Func هو مفوض والمفوضات هي مؤشرات للوظائف فكيف يقوم المترجم إذا بالعمل؟ في هذه الحالة يقوم المترجم بإصدار وظيفة جديدة ويربطها بمفوض يشير إلى تلك الوظيفة الجديدة

```
Private Function $GeneratedFunction$(x As Integer) As Integer
    Return x * 2
End Function

Sub TestLambda()
    Dim doubleIt As Func(Of Integer, Integer) = _
        AddressOf $GeneratedFunction$
    Console.WriteLine(doubleIt(10)
End Sub
```

حيث يأخذ المترجم تعبير لمدا وينشئ وظيفة جديدة بمحتوياته ويغير عبارة التصريح بحيث يأخذ تعبير لمدا عنوان الوظيفة الجديدة المولدة ففي هذه الحالة يتم توليد الوظيفة بنفس الأب الذي يحتوي على الطريقة التي تستخدم تعبير لمدا فإن كان TestLambda معرف في الفئة C فسوف يتم تعريف الوظيفة الجديدة في الفئة C أيضا ونلاحظ أن هذه الوظيفة غير قابلة للاستدعاء ويتم التصريح عنها باستخدام محدد الوصول Private

تعابير لمدا ورفع المتغيرات

في الأمثلة السابقة يشير جسم تعابير لمدا إلى متغيرات يتم تمريرها إلى تلك المتغيرات ومع ذلك تأتي قوة تعابير لمدا مع ثمار رفع المتغيرات وجميع تعابير لمدا مبنية على مبدأ متشابه. وتعبير لمدا يمكن أن يستخدم متغيرات مرتبطة أو متغيرات حرة لم يتم تعريفها ضمن التوقيع الخاص بتعبير لمدا فالمتغيرات الحرة ممكن أن يكون قد تم التصريح عنها في الإجراء المستدعي للتعبير فقد تكون متغيرات محلية أو محددات ممررة لذلك الإجراء والتعابير المرتبطة تكون تلك التي تم التصريح عنها في جسم التعبير أو عناصر في الفئة المحتوية للتعبير لمدا متضمنا الفئة الأب لتلك الفئة. وهذا هام من أجل التمييز بين المتغيرات المرتبطة والحرة في تعابير لمدا الخاصة بك لأنها تؤثر على دلالة تعبير لمدا والكود الذي يتم توليده وبالتالي يؤثر على صحة برنامجك وهذا مثال يحتوي على تعابير لمدا تستخدم متغيرات مرتبطة وأخرى حرة

```
Function MakeLambda() As Func(Of Integer, Integer)
    Dim y As Integer = 10
    Dim addTen As Func(Of Integer, Integer) = Function(ByVal x) x + y
    Return addTen
End Function

Sub UseLambda()
    Dim addTen = MakeLambda()
    Consloe.WriteLine(addTen(5))
End Sub
```

فهذا الكود سيقوم بطباعة 15 على نافذة الكونسول عندما يتم استدعاء UseLambda ولكن يمكن أن تسأل نفسك كيف يعمل هذا؟ تحدد الوظيفة MakeLambda المتغير y كمتغير محلي والتعبير لمدا يستخدم y ولكن التعبير لمدا يتم إعادته كنوع معاد من الوظيفة MakeLambda والوظيفة UseLambda تحصل على التعبير لمدا من الوظيفة MakeLambda وتنفذ التعبير لمدا ويبدو الأمر كما لو أن المتغير y قد تم تذكره من قبل التعبير لمدا. ففترة حياة المتغير y تنتهي مع نهاية الطريقة MakeLambda فعندما نحصل على التعبير لمدا من MakeLambda فسوف تصبح MakeLambda خارج المجال ويجب إزالة المساحة التي تحجزها في المكدس وبطريقة ما يعلق هذا المتغير مع تعبير لمدا وهذا ما يعرف برفع المتغير Variable Lifting ففي هذه الحالة يدعى المتغير y بالمتغير المرفوع وكما ترى فالمتغيرات المرفوعة تعتبر ميزة برمجية قوية فالمترجم يقوم بالكثير من العمل من أجل تمكينك من إمساك حالة المتغير حيث يحفظها خارج مجال فترة حياتها الطبيعية فعندما يصادف المترجم تعابير لمدا تستخدم متغيرات حرة يقوم برفع المتغير إلى فئة تدعى Closure بحيث تكون فترة حياة هذه الفئة تمتد إلى ما بعد فترة حياة المتغيرات الحرة المستضافة داخلها ويقوم المترجم بإعادة كتابة الوصول إلى المتغيرات في الطرق ليتم الوصول إلى نسختها الموجودة في الفئة Closure

دعنا نسير مرة أخرى عبر المثال MakeLambda

```
Dim MakeLambda() As Func(Of Integer, Integer)
    Dim y As Integer = 10
    Dim addTen As Func(Of Integer, Integer) = Function(ByVal x) x + y
        Return addTen
End Function
```

وكما قمنا بالتحليل سابقا فالمتغير x مرتبط بمحدد التعبير لمدا ولكن المتغير y تعبير حر ويقوم المترجم بالكشف عن ذلك ويتابع بإنشاء الفئة Closure التي تلتقط المتغيرات الحرة كما في تعريف تعبير لمدا

```
Public Class _Closure$__1
    Public y As Integer
    Public Function _Lambda$__1(ByVal x As Integer) As Integer
        Return x + Me.y
    End Function
End Class
```

يمكنك رؤية أن متغير Closure يلتقط المتغير y ويخزنه في الفئة Closure ويتم تحويل المتغير الحر بعدها إلى متغير مرتبط داخل الفئة Closure كما يقوم المترجم بإعادة كتابة الطريقة التي تحتوي على التعبير لمدا لتبدو كما يلي

```
Function MakeLambda() As Func(Of Integer, Integer)
    Dim Closure As New _Closure$__1
    Closure.y = 10
    Return AddressOf Closure._Lambda$__1
End Function
```

يمكنك الآن رؤية كيف يقوم المترجم بإنشاء المتغير Closure ويعيد كتابة المتغير y الذي تم رفعه ضمن المتغير Closure ويضبط قيمته ويعيد ببساطة عنوان تعبير لمدا المخزن ضمن الفئة Closure ومن الهام ملاحظة أن المترجم يقوم برفع المتغيرات الحرة في تعابير لمدا فقط ويتم التقاط حالة المتغير في Closure الذي يبقى موجودا طالما أن تعبير لمدا بقي موجودا. انظر للمثال التالي

```
Sub Test()
    Dim y As Integer = 10
    Dim Lambda As Func(Of Integer, Integer) = Function(ByVal x) x + y
    y = 20
    Console.WriteLine(Lambda(5))
End Sub
```

ما هي القيمة التي تظهر عند تنفيذ الوظيفة السابقة؟ إن قلت 25 فقد أصبت. فلماذا 25 إذا؟ المترجم يقوم بالتقاط وإعادة كتابة جميع المتغيرات الحرة y إلى نسخة Closure كالتالي

```
Sub Test()
    Dim Closure As New $Closure_Compiler_Generated_Name$
    Closure.y = 10
    Dim Lambda = AddressOf Closure.Lambda_1
    Closure.y = 20
    Console.WriteLine(Lambda(5))
End Sub
```

ففي الوقت الذي يتم تنفيذ تعبير لمدا فيه تكون قيمة y قد تغيرت إلى 20 وبهذا فعندما يتم تنفيذ تعبير لمدا يعيد 20 + 5 وهذا هام جدا لأنه عندما نأتي للحديث عن الحلقات وأن المتغيرات الحرة يتم التقاطها في Closure وحيد قد ترى تصرفات غريبة. انظر للمثال التالي

```
Sub Test()
    For I = 1 To 5
        StartThread(Function() I + 10)
    Next
```

```
End Sub
```

افرض أن StartThread ينشئ مسارا جديدا ويطبع النتيجة على الكونسول وطالما أنه تم التقاطه إلى Closure فيمكن أن تكون الحلقة قد غيرت قيمة I في الوقت الذي يقوم المسار فيه باستدعاء تعبير لمدا وفي هذه الحالة فالبرنامج قد لا يطبع النتيجة المتوقعة وبدلا عن ذلك عليك رؤية المتغير الملتقط داخل الحلقة

```
Sub Test
    For I = 1 To 5
        Dim x = I
        StartThread(Function() x + 10)
    Next
End Sub
```

فالكود سيلتقط الآن قيمة x في Closure والبرنامج سيطبع القيم كما هو متوقع ومن الهام جدا معرفة أية متغيرات سيتم رفعها عندما سيتم تنفيذ تعبير لمدا ومتى سيتم تغيير قيمة تلك المتغيرات المرفوعة وبذلك يمكنك التأكد من أن برنامجك يتم تنفيذه بصورة صحيحة.

استخدام تعابير لمدا بالشكل الأمثل

في فيجول بايزيك 2008 يمكنك تمرير تعبير واحد كجسم لتعبير لمدا وقد تم تقديم كلمة محجوزة ثلاثية جديدة هي If لتمكنك من كتابة تعابير شرطية ذات نوع كامل

```
Dim x = IF(condition, 10, 20)
```

والكلمة المحجوزة If مشابهة لاستدعاء الوظيفة IIF فيما عدا أنها آمنة ضد النوع. وهذا يعني أنه في المثال السابق يتتبع المترجم كلا فرعي الكلمة المحجوز If ويعيد Integer وبهذا فهو يطبق قواعد الاستدلال على النوع ويقرر أن نوع x هو integer ولكن استخدام IIf سيعيد النوع Object. كما يمكنك استخدام If في تعبير لمدا

```
Dim x = Function(c As Customer) _
    If(c.Age >= 18, c.Address, c.Parent.Address)
```

ففي المثال السابق افترض أنه لديك فئة Customer يتضمن تعريفها الخاصية Address التي تمثل العنوان الحالي للزبون حيث أن تعبير لمدا يستخدم التعبير الثلاثي Ternary Expression لتطبيق الشرط على محدد الدخل فإن كان عمر الزبون مساويا أو أكثر من 18 فهو يعيد عنوانه وإلا فهو يعيد عنوان والده وهنا يتم استخدام الاستدلال على النوع أيضا ويقوم المترجم بتحديد نوع تعبير لمدا ليكون Address ثم يقوم بإنشاء النوع المفوض x بالطريقة التي تمت مناقشتها سابقا حيث يأخذ النوع المفوض Customer كدخل ويعيد Address.

Nullable Value Types

في بعض الأحيان نتعامل مع قيمة ذات نوع ولكنها قد لا تملك قيمة محددة في حالات معينة فحقل في قاعدة بيانات مثلا يمكن تمييزه بين أن له قيمة ذات معنى أو أن ليس له قيمة أبدا. عندها يمكننا توسيع أنواع القيم لتأخذ إما قيما عادية أو قيم لا شئ null Value وهذا التوسيع يدعى nullable type.

كل nullable type يتم إنشاؤه من التركيب Nullable(T). ففي المثال التالي يتم التصريح عن متغير يحمل النوع Nullable Boolean Type كما يلي

```
Dim ridesBusToWork1? As Boolean
Dim ridesBusToWork2 As Boolean?
Dim ridesBusToWork3 As Nullable(Of Boolean)
```

فالمتغير ridesBusToWork يمكن أن يحمل القيمة True أو القيمة False أو أن لا يحمل أي قيمة إطلاقا وتكون القيمة الافتراضية له هي أنه لا يحمل أي قيمة وفي هذه الحالة قد يعني أنه لم يتم الحصول على تلك المعلومة من ذلك الشخص وفي المقابل القيمة False قد تعني أن الشخص لا يركب الباص للعمل.

يمكنك التصريح عن متغيرات أو خصائص أو حتى مصفوفات أو إجراءات من nullable types كما يمكنك إعادة Nullable type من وظيفة ما. ولكن لا يمكنك إنشاء nullable type من نوع مرجعي Reference type مثل المصفوفات أو الفئات أو String فالنوع الأساسي يجب أن يكون نوع بالقيمة Value type

تعتبر الخصائص HasValue و Value هي العناصر الأكثر أهمية في nullable type فمن أجل متغير من النوع nullable type تخبرنا الخاصية HasValue فيما إذا كان للمتغير قيمة محددة أم لا فإن كانت قيمة تلك الخاصية True عندها يمكنك قراءة قيمة المغير من الخاصية Value ويجب عليك الانتباه إلى أن كلتا الخاصيتين HasValue و Value هما خاصيتين للقراءة فقط.

عندما تصرح عن متغير من النوع nullable type تكون القيمة الافتراضية للخاصية HasValue هي False وهذا يعني أن المتغير في الحالة الافتراضية لا يحمل أي قيمة محددة عوضا عن القيمة الافتراضية لنوع القيمة المؤسس لهذا النوع ففي المثال التالي لا يكون للمتغير numberOfChildren قيمة محددة مع أن القيمة الافتراضية للنوع Integer هي الصفر

```
Dim numberOfChildren? As Integer
```

وتكون القيمة لا شئ Null Value مفيدة لتوضيح أن قيمة المتغير غير معروفة أو غير محددة فإن تم التصريح عن المتغير numberOfChildren على أنه Integer فلن يكون هناك قيمة تشير إلى أنه لم يتم توفير المعلومات المطلوبة بعد

كما يمكن ضبط قيمة المتغير من النوع nullable type بالطريقة الاعتيادية كما في المثال التالي الذي يضبط قيمة للمتغير numberOfChildren الذي تم التصريح عنه في المثال السابق

```
numberOfChildren = 2
```

وإن كانت لمتغير أو خاصية من النوع nullable type قيمة محددة يمكنك إعادتها للقيمة الأساسية بعدم احتوائها على قيمة وذلك بضبطها إلى Nothing كما في المثال

```
numberOfChildren = Nothing
```

مع انه يمكنك ضبط القيمة Nothing للمتغير من النوع Nullable type إلا أنه لا يمكنك فحصه بالمقارنة مع Nothing باستخدام علامة المساواة وتكون قيمة المقارنة التي تستخدم علامة المساواة مثل someVar = Nothing دائما مساوية لـ Nothing. بدلا عن ذلك يمكنك فحص قيمة الخاصية hasValue من اجل القيمة False أو باستخدام المعامل Is أو المعامل IsNot

للحصول على القيمة المخزنة في متغير من النوع nullable type يجب عليك أولا فحص الخاصية HasValue للتأكد من أنها تحمل قيمة فإن حاولت قراءة قيمة ذلك المتغير وكانت قيمة خاصيته HasValue مساوية لـ False سوف يقوم فيجول بايزيك بإطلاق استثناء InvalidOperationException ويرينا المثال التالي الطريقة المنصوح بها لقراءة قيمة المتغير numberOfChildren الخاصة بالمثال السابق

```
If numberOfChildren.HasValue Then
    MsgBox("There are " & CStr(numberOfChildren) & " children.")
Else
    MsgBox("It is not known how many children there are.")
End If
```

عندما يتم استخدام متغيرات من النوع Nullable Boolean في التعابير المنطقية يمكن أن تكون القيمة True أو False أو Nothing وفيما يلي جدول الحقيقة من أجل And و Or بما أن b1 و b2 يملكان ثلاثة قيم محتملة يكون هناك تسعة احتمالات للمقارنة

b1	b2	b1 And b2	b1 Or b2
Nothing	Nothing	Nothing	Nothing
Nothing	True	Nothing	True
Nothing	False	False	Nothing
True	Nothing	Nothing	True
True	True	True	True
True	False	False	True
False	Nothing	False	Nothing
False	True	False	True
False	False	False	False

عندما تكون قيمة المتغير المنطقي أو التعبير Nothing فالقيمة هي ليست True و ليست False أيضا انظر للمثال التالي

```
Dim b1? As Boolean
Dim b2? As Boolean
b1 = True
b2 = Nothing

' The following If statement displays "Expression is not true".
If (b1 And b2) Then
    Console.WriteLine("Expression is true")
Else
    Console.WriteLine("Expression is not true")
End If

' The following If statement displays "Expression is not false".
If Not (b1 And b2) Then
    Console.WriteLine("Expression is false")
Else
    Console.WriteLine("Expression is not false")
End If
```

ففي هذا المثال تقيم b1 And b2 إلى Nothing وبالنتيجة يتم تنفيذ قسم Else في كلا تعبير If ويكون الخرج كما يلي

```
Expression is not true
Expression is not false
```

لاحظ ان AndAlso و OrElse اللتين تستخدمان التقييم المختصر تقومان بتقييم معاملهما الثاني في حالة كون التعبير الأول قد تم تقييم قيمته إلى Nothing

إن كان كلا أو أحد المعاملات في معادلة رياضية أو منطقية أو إزاحة nullable فستكون النتيجة أيضا nullable وإن كان لكلا المعاملين قيمة لا تساوي Nothing تتم العملية وفق قيم تلك المعاملات كما لو أنهما من نوع قيمة وليس nullable ففي المثال التالي المتغير compare1 والمتغير sum1 نوعان ضمنيان فإن أوقفت مشيرة الفأرة قليلا فوقهما ستلاحظ أن المترجم يشير إلى أن كلاهما من nullable type

```
'Variable n is a nullable type, but both m and n have proper values.
Dim m As Integer = 3
Dim n? As Integer = 2

' The comparison evaluated is 3>2, but compare1 is inferred to be of
' type Boolean?.
Dim compare1 = m > n
' The values summed are 3 and 2, but sum1 is inferred to be of type Integer?.
Dim sum1 = m + n

' The following line displays: 3 * 2 * 5 * True
Console.WriteLine(m & " * " & n & " * " & sum1 & " * " & compare1)
```

وإن كانت قيمة واحد أو أكثر من المعاملات Nothing فالنتيجة ستكون Nothing

```
' Change the value of n to Nothing.
n = Nothing

Dim compare2 = m > n
Dim sum2 = m + n

' Because the values of n, compare2, and sum2 are all Nothing, the
' following line displays 3 * * *
Console.WriteLine(m & " * " & n & " * " & sum2 & " * " & compare2)
```

تشكل قواعد البيانات أكثر أماكن استخدام nullable types أهمية مع أن ليست جميع أغراض قواعد البيانات تدعم nullable types ولكن الـ TableAdapter المولد من قبل المصمم يدعمها

Object Initializers

تمكنك Object Initializers من تحديد خصائص غرض معقد ضمن تعبير واحد وتستخدم لتعريف متغيرات من كلا من الأنواع المعروفة والمجهولة فلو فرضنا أنه لدينا فئة بسيطة Employee معرفة على الشكل

```
Public Class Employee
    Private _name As String
    Private _Salalry As Short
    Private _Address As String

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property

    Public Property Salary() As Short
        Get
            Return _Salalry
        End Get
        Set(ByVal value As Short)
            If value > 0 Then
                _Salalry = value
            End If
        End Set
    End Property

    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property
End Class
```

يمكننا باستخدام تعريف متغير يشير إلى تلك الفئة واسندا الخصائص كما في الكود التالي مع أننا لسنا مضطرين هنا لضبط قيم كافة الخصائص التي تحتويها الفئة فنقوم بضبط قيم الخصائص التي نحتاج لضبطها فقط

```
Dim Empl3 = New Employee With {.Name = "Mazen", .Salary = 8500}

Dim Empl1 As New Employee With {.Name = "Reem", .Salary = 10000}
```

كما يمكننا اختصار قسم AS هنا فيمكن كتابة التصريح كما يلي وذلك اعتماد على local type inference

```
Dim Empl5 = New Employee With {.Name = "Ahmad"}
```

بينما كنا في السابق وباستخدام نفس الفئة كما يلي

```
Dim Empl2 As New Employee
With Empl2
```

```
        .Name = "Ahamd"
        .Salary = 11500
    End With
```

وإن كانت لدينا فئة تحتاج لتمرير قيم لمشيد الفئة مثل الفئة Person مثلا فيمكننا أيضا استخدام نفس الطريقة لضبط خصائص أخرى لا يتم تمريرها لمشيد الفئة

```
Dim Per1 As New Person("Ghassan") With {.Address = "Damas"}
```

كما تستخدم هذه الطريقة أيضا لتعريف الأنواع المجهولة

```
Dim Visitor = New With {.Name = "Mussa", .Account = 232536}
```

وكما نلاحظ من طريقة التعريف فصيغة تعريف الأنواع المعروفة مماثلة في الشكل للأنواع المجهولة ففي الأنواع المعروفة لاحظ وجود اسم الفئة بعد الكلمة new بينما عندما نعرف نوعا مجهولا لايوجد اسم للفئة بعد الكلمة new بسبب أن الأنواع المجهولة ليس لها اسم فئة قابلة للاستخدام فعند استخدام فئة معروفة عند التصريح يجب أن تكون الخصائص التي نريد ضبط قيمها موجودة فعلا والتصريح ينشئ متغيرا يشير إلى تلك الفئة ومن أجل تعريف النوع المجهول يقوم المترجم بإنشاء فئة جديدة لذلك المتغير تحتوي الخصائص المشار إليها في التصريح ويحدد اسمها عند الترجمة وقد يختلف لاسم من عملية ترجمة لأخرى لذلك لا يمكن الاعتماد على اسم الفئات المجهولة ضمن الكود أو التعريف

وإليك بعض الملاحظات الخاصة بالتعريف

- قائمة التعريف بعد With لا يمكن أن تكون فارغة

- لا يمكن تكرار تعريف قيمة لخاصية أكثر من مرة في نفس التعريف

- يمكن ضبط قيمة خاصية من خاصية أخرى

- في حال كانت إحدى الخصائص فئة يمكن تعشيش التصريح بنفس الطريقة

```
Dim cust12 = New Customer With {.Name = "Toni Poe", _
                                .Address = New AddressClass _
                                With {.City = "Louisville", _
                                      .State = "Kentucky"}}
Console.WriteLine(cust12.Address.State)
```

- لا يمكن استخدام عناصر مشتركة Shared أو للقراءة فقط ReadOnly أو الثوابت أو استدعاء الطرق في القائمة بعد كلمة With

- لا يمكن استخدام الخصائص التي تمتلك فهرسا أو المشروطة كمصفوفة مثلا فالتعريفات التالية مثلا غير صحيحة

```
'' Not valid.
' Dim c1 = New Customer With {.OrderNumbers(0) = 148662}
' Dim c2 = New Customer with {.Address.City = "Springfield"}
```

الاستدلال المحلي على النوع Local Type Inference

يستخدم المترجم في فيجول بايزيك 2008 الاستدلال على النوع Type Inference لتحديد نوع المتغيرات المحلية التي تم التصريح عنها بدون استخدام فقرة As في تعبير التصريح حيث يستدل المترجم على نوع المتغير من نوع التعبير الذي يضبط قيمة ذلك المتغير مما يوفر إمكانية تعريف المغيرات بدون تحديد نوعها كما في المثال التالي

```
Public Sub inferenceExample()

    ' Using explicit typing .
    Dim num1 As Integer = 3

    ' Using local type inference.
    Dim num2 = 3

End Sub
```

ولا يمكن استخدام الاستدلال على النوع عند تعريف الحقول في الفئة Class Fields فإن كان num2 في المثال السابق حقلا في فئة بدلا عن كونه متغيرا محليا فسوف يولد التصريح خطأ إذا كان Option Strict On وسوف يصنف num2 على أنه غرض Object إن كان Option Strict Off وبشكل مشابه فنوع المتغيرات الساكنة Static Variables لا يمكن الاستدلال عليها إن كان Option Strict On وإن كان Option Strict Off فنوع المتغير الساكن سيكون غرض Object فإن لم تكن تريد من المتغير num2 في المثال السابق أن يكون من النوع Integer فيمكنك تحديد نوعا آخر عند التصريح عنه

Dim num3 As Object = 3 or Dim num4 As Double = 3

والكود الذي يستخدم استدلال النوع يشابه الكود الذي يعتمد على الربط المتأخر Late Binding الذي سيكون نوعه معروفا فقط في زمن التشغيل. ومعرفة النوع بشكل مبكر يمكن المترجم من تحديد المشاكل قبل التنفيذ وحجز الذاكرة بدقة وإجراء عمليات التحسين الأخرى بالإضافة إلى تمكين بيئة التطوير من تزويد المبرمج بـ IntelliSense والمساعدة حول أعضاء ذلك الغرض بالإضافة إلى تفضيله لاعتبارات خاصة بالأداء بسبب أن جميع البيانات التي تخزن باستخدام الربط المتأخر يجب تغليفها وكأنها من النوع Object والوصول إلى الأعضاء في زمن التشغيل سيكون أبطأ.

يحدث الاستدلال على النوع عندما يتم التصريح عن المتغير بدون استخدام فقرة As في تعبير التصريح وضبط قيمة لذلك المتغير فيستخدم المترجم نوع تلك القيمة كنوع للمتغير فمثلا سطور الكود التالية تعرف متغيرا من النوع String

```
' Using explicit typing.
Dim name1 As String = "Springfield"

' Using local type inference.
Dim name2 = "Springfield"
```

ويستعرض الكود التالي طريقتان متكافئتان لإنشاء مصفوفة من النوع Integer

```
' Using explicit typing.
Dim someNumbers1() As Integer = New Integer() {4, 18, 11, 9, 8, 0, 5}

' Using local type inference.
Dim someNumbers2 = New Integer() {4, 18, 11, 9, 8, 0, 5}
```

كما يمكنك استخدام الاستدلال على النوع لتحديد نوع متغير التحكم لحلقة تكرارية ففي الكود التالي سيتعرف المترجم على num بأنه من النوع Integer لأن someNumbers2 عبارة عن مصفوفة Integer

```
Dim total = 0
For Each number In someNumbers2
    total += number
Next
```

ويستخدم الاستدلال على النوع في العبارة Using أيضا لتحديد نوع اسم المصدر كما هو واضح في المثال التالي

```
Using proc = New System.Diagnostics.Process
    ' Insert code to work with the resource.
End Using
```

ويستدل على نوع المتغير من القيمة المعادة من الإجراء أيضا كما هو ظاهر في الكود التالي حيث يكون pList1 و pList2 عبارة عن Lists of Processes

```
' Using explicit typing.
Dim pList1() As Process = Process.GetProcesses()

' Using local type inference.
Dim pList2 = Process.GetProcesses()
```

وقد قدم فيجول بايزيك 2008 خيارا جديدا هو Option Infer يمكنك من تحديد إذا كان الاستدلال المحلي على النوع مسموحا أم لا في ملف معين. فلتمكين أو تعطيل خيار الاستدلال على النوع اكتب التعبير المناسب من السطرين التاليين في بداية الملف

```
Option Infer On
Option Infer Off
```

وإن لم تقم بتحديد قيمة للخيار Option Infer في الكود فالمترجم سيستخدم الخيار الافتراضي Option Infer On من أجل المشاريع التي تم إنشاؤها في Visual Basic 2008 والخيار Option Infer Off من أجل المشاريع التي تمت ترقيتها من إصدارات سابقة. وإن تضاربت قيمة الخيار Option Infer في الملف مع القيمة المضبوطة في خيارات بيئة التطوير أو في سطر الأوامر فسوف يتم استخدام القيمة الموجودة في الملف.

ويستخدم الاستدلال على النوع فقط في المتغيرات الغير ساكنة Non-Static ولا يمكن استخدامها في تعريف حقول الفئة Class Fields أو الخصائص Properties أو الإجراءات Functions

إجبار المستخدم على اختيار واحدة من عدة قيم محددة سابقا في صندوق النصوص

تقوم الفكرة على استخدام الحدث Validating للتأكد من أن المستخدم قد اختار ما يطابق قائمة القيم المقترحة.

للتجربة قم بإنشاء مشروع جديد ثم أضف إليه صندوقي نصوص

في الحدث Load للنموذج ضع الكود التالي

```
' تعريف مصفوفة ستحمل خياراتنا للإكمال التلقائي
Dim Lis As New List(Of String)
Lis.Add("Visual Basic")
Lis.Add("Visual C#")
Lis.Add("Visual C++")
Lis.Add("Pascal")
Lis.Add("Delphi")
Lis.Add("C++")
Lis.Add("2100")
Lis.Add("3200")
Lis.Add("Nokia 6600")
Lis.Add("Nokia 3250")
Lis.Add("Nokia 7610")

' تحديد طريقة الإكمال التلقائي
Me.TextBox1.AutoCompleteMode = AutoCompleteMode.SuggestAppend
' تحديد مصدر الإكمال التلقائي
Me.TextBox1.AutoCompleteSource = AutoCompleteSource.CustomSource

' إضافة القائمة الخاصة بنا كمصدر للإكمال التلقائي
Me.TextBox1.AutoCompleteCustomSource.Clear()
Me.TextBox1.AutoCompleteCustomSource.AddRange(Lis.ToArray)
```

في الحدث Validating لصندوق النصوص TextBox1 ضع الكود التالي

```
If Me.TextBox1.AutoCompleteCustomSource.Count = 0 Then
    ' إذا كانت قائمة الإكمال التلقائي فارغة
    Me.TextBox1.Text = String.Empty
    Exit Sub
Else
    ' تحديد إذا كان النص موجود ضمن قائمة الإكمال التلقائي
    For Each Str As String In Me.TextBox1.AutoCompleteCustomSource
        If Str.ToUpper = Me.TextBox1.Text.ToUpper Then
            Me.TextBox1.Text = Str
            Exit Sub
        End If
    Next
End If
' إذا وصلنا إلى هنا فهذا يعني أن النص المدخل غير موجود
' في قائمة الإكمال التلقائي فتكون عملية التحقق فاشلة
e.Cancel = True
```

شغل البرنامج وقم بالتجربة

استخدام الوظائف المخزنة Using Stored Procedures

في العديد من الحالات البرمجية بدلا من تنفيذ عبارة SQL مباشرة ستحتاج إلى تنفيذ ما يدعى بالوظائف المخزنة أو Stored Procedures والتي تعتبر بدورها طريقة ممتازة لتغليف منطق قواعد البيانات وتعزيز الأمان في التطبيقات متعددة الطبقات multitiered ولتنفيذ وظيفة مخزنة ستحتاج إلى Command object تقوم بضبط خاصيته CommandType إلى StoredProcedure وخاصية CommandText إلى اسم الوظيفة المخزنة وفي المثال التالي نقوم بتنفيذ وظيفة مخزنة تقرأ عدد employees في قاعدة البيانات Pubs

```
Public Overloads Function CountEmployees() As Integer
    Dim oPubConnection As New SqlConnection
    Dim sConnString As String
    Dim oSqlCommand As SqlCommand
    Try
        sConnString = "Data Source=drcsrv01;Initial Catalog=pubs;" & _
        "Integrated Security=True"
        oPubConnection.ConnectionString = sConnString
        oPubConnection.Open()
        oSqlCommand = New SqlCommand()
        oSqlCommand.Connection = oPubConnection
        oSqlCommand.CommandText = "GetEmployeeCount"
        oSqlCommand.CommandType = CommandType.StoredProcedure
        Return oSqlCommand.ExecuteScalar
    Catch oEx As Exception
        MessageBox.Show(oEx.Message)
    Finally
    If Not oPubConnection Is Nothing Then
        oPubConnection.Close()
    End If
End Try
```

في العادة عندما نقوم بتنفيذ وظيفة مخزنة نحتاج إلى تمرير وسائط دخل Input Parameters وقد تستعيد نتائج الخرج من خلال وسائط الخرج Output Parameters وللتعامل مع الوسائط يجب علينا تعريف مرجع لـ SqlParameter وتزويده باسم الوسيطة ونوعها وبالنسبة لبعض أنواع البيانات ستحتاج لتمرير حجم هذه البيانات أيضا وعندما تمرر وسيطة خرج، دخل، دخل-خرج عليك أيضا أن تحدد اتجاهها في المثال التالي نمرر وسيطة دخل عبارة عن حرف وتعيد عدد الـموظفين الذين تبدأ أسماء عائلاتهم بذلك الحرف والعدد يعاد بشكل وسيطة خرج

```
Public Overloads Function CountEmployees(ByVal LInitial As String) As Integer
    Dim oPubConnection As New SqlConnection
    Dim sConnString As String
    Dim oSqlCommand As SqlCommand
    Try
        sConnString = "Data Source=drcsrv01;Initial Catalog=pubs;" & _
        "Integrated Security=True"
        oPubConnection.ConnectionString = sConnString
        oPubConnection = New SqlConnection(sConnString)
        oPubConnection.Open()
        oSqlCommand = New SqlCommand()
        oSqlCommand.Connection = oPubConnection
        oSqlCommand.CommandText = "GetEmployeeCountbyLInitial"
        oSqlCommand.CommandType = CommandType.StoredProcedure
        Dim oInputParam As SqlParameter = _
        oSqlCommand.Parameters.Add("@LInitial", SqlDbType.Char, 1)
        oInputParam.Value = LInitial
        Dim oOutPutParam As SqlParameter = _
        oSqlCommand.Parameters.Add("@EmployeeCount", SqlDbType.Int)
```

```
        oOutPutParam.Direction = ParameterDirection.Output
        oSqlCommand.ExecuteNonQuery()
        Return oOutPutParam.Value
    Catch oEx As Exception
        MessageBox.Show(oEx.Message)
    Finally
       If Not oPubConnection Is Nothing Then
           oPubConnection.Close()
       End If
    End Try
End Function
```

الأنواع المجهولة Anonymous Types

يقدم فيجول ستوديو 2008 الأنواع المجهولة anonymous types والتي تمكنك من إنشاء الأغراض Objects بدون كتابة تعريف فئة Class definition من أجل نوع البيانات وعوضا عن ذلك يولد المترجم الفئة من أجلك ولن يكون للفئة اسما قابلا للاستخدام حيث تكون هذه الفئات موروثة مباشرة من Object وتمتلك الخصائص التي تحددها عند تعريف الغرض Object وبما أن نوع البيانات لم يتم تحديده يتم الإشارة إليه على أنه نوع مجهول anonymous type. حيث يصرح المثال التالي عن المتغير product كمتغير من النوع anonymous type ممتلكا الخاصيتين Name و Price

```
' Variable product is an instance of a simple anonymous type.

Dim product = New With {Key .Name = "paperclips", .Price = 1.29}
```

حيث يستخدم تعبير الاستعلام التالي الأنواع المجهولة لدمج أعمدة البيانات المحددة بواسطة الاستعلام وبما أنه لا يمكنك تحديد نوع النتيجة مقدما بسبب عدم إمكانية التنبؤ بالأعمدة التي يمكن أن يختارها استعلام معين فتمكنك الأنواع المجهولة من كتابة استعلام يختار عدد من الأعمدة بأي ترتيب نريده فيقوم المترجم بإنشاء نوع البيانات المماثل لتلك الخصائص المحددة بذلك الترتيب المعين. وفي المثال التالي يكون Products عبارة عن قائمة من أغراض Product وكل منها يمتلك خصائص عديدة بحيث يحمل المتغير namePriceQuery تعريف الاستعلام الذي يعيد عند تنفيذه مجموعة من الأنواع المجهولة التي تمتلك الخاصيتين Name و Price

```
Dim namePriceQuery = From prod In products_

                   Select prod.Name, prod.Price
```

والمتغير nameQuantityQuery يحمل تعريف الاستعلام الذي يعيد عند تنفيذه مجموعة من الأنواع المجهولة التي تمتلك خاصيتين Name و OnHand

```
Dim nameQuantityQuery = From prod In products_

                     Select prod.Name, prod.OnHand
```

تعريف نوع مجهول Declaring an Anonymous Type

تعريف متغير من نوع مجهول يستخدم قائمة بناء لتحديد خصائص ذلك النوع بحيث يمكنك تحديد هذه الخصائص فقط عند الإعلان عن النوع المجهول ولا يمكن استخدام بقية عناصر الفئات مثل الطرائق والأحداث في الأنواع المجهولة ففي المثال التالي يكون Product1 من نوع مجهول يمتلك خاصيتين Name و Price

```
'Variable product1 is an instance of a simple anonymous type.

Dim product1 = New With {.Name = "paperclips", .Price = 1.29{

- 'or-

'product2 is an instance of an anonymous type with key properties.

Dim product2 = New With {Key .Name = "paperclips", Key .Price = 1.29{
```

فإن قمت بتحديد الخصائص كخصائص مفتاحية key properties أصبح بإمكانك استخدامها لمقارنة نوعين مجهولين هل هما متساويين أم لا ومع ذلك فقيم الخصائص المفتاحية لا يمكن تغييرها فهي للقراءة فقط. مع ملاحظة أن التصريح عن نوع مجهول يماثل التصريح عن نوع مسمى باستخدام باني الغرض object initializer

```
'Variable product3 is an instance of a class named Product.
```

Dim product3 = New Product With {.Name = "paperclips", .Price = 1.29{

الخصائص المفتاحية Key Properties

تختلف الخصائص المفتاحية عن العادية بعدة أمور:

- تستخدم الخصائص المفتاحية فقط لمقارنة المساواة بين نوعين مجهولين
- لا يمكن تغيير قيم الخصائص المفتاحية فهي دائما للقراءة فقط
- فقط الخصائص المفتاحية يتم تضمينها ضمن الـ Hash Code الذي يولده المترجم من أجل الأنواع المجهولة

المساواة Equality

تكون متغيرات الأنواع المجهولة متساوية عندما تكون متغيرات لنفس النوع المجهول ويقوم المعالج بمعاملة متغيرين كمتغيرين من نفس النوع إذا توفرت فيهما الشروط التالية

- تم التصريح عنهما في نفس المجمع
- تمتلك خصائصهما نفس الاسم والنوع وتم التصريح عنها بنفس الترتيب وتكون مقارنة الأسماء غير حساسة لحالة الأحرف
- نفس الخصائص فيها محددة كخصائص أساسية
- يمتلك كل نوع خاصية أساسية واحدة على الأقل

والتصريح عن نوع مجهول الذي لا يمتلك أي خاصية مفتاحية يكون مساويا لنفسه فقط

```
'prod1 and prod2 have no key values.

Dim prod1 = New With {.Name = "paperclips", .Price = 1.29}

Dim prod2 = New With {.Name = "paperclips", .Price = 1.29}


'The following line displays False, because prod1 and prod2 have no

'key properties.

Console.WriteLine(prod1.Equals(prod2))


'The following statement displays True because prod1 is equal to itself.

Console.WriteLine(prod1.Equals(prod1))
```

وتكون قيمة متغيرين لنفس النوع المجهول متساويين إذا كانت قيمة خصائصهما المفتاحية متساوية كما في المثال التالي الذي يوضح كيفية فحص هذه المساواة

```
Dim prod3 = New With {Key .Name = "paperclips", Key .Price = 1.29}

Dim prod4 = New With {Key .Name = "paperclips", Key .Price = 1.29}

'The following line displays True, because prod3 and prod4 are

'instances of the same anonymous type, and the values of their

'key properties are equal.

Console.WriteLine(prod3.Equals(prod4))


Dim prod5 = New With {Key .Name = "paperclips", Key .Price = 1.29}
```

```
Dim prod6 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 423}

 'The following line displays False, because prod5 and prod6 do not

 'have the same properties.

Console.WriteLine(prod5.Equals(prod6))


Dim prod7 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 24}

Dim prod8 = New With {Key .Name = "paperclips", Key .Price = 1.29, OnHand = 423}

 'The following line displays True, because prod7 and prod8 are

 'instances of the same anonymous type, and the values of their

 'key properties are equal. The equality check does not compare the

 'values of the non-key field.

Console.WriteLine(prod7.Equals(prod8))
```

القيم القابلة للقراءة فقط

لا يمكن تغيير قيم الخصائص المفتاحية فمثلا في prod8 في المثال السابق الحقول Name و Price قابلة للقراءة فقط في حين أن الحقل OnHamd يمكن تغيير قيمته

```
 'The following statement will not compile, because Name is a key

 'property and its value cannot be changed.

 'prod8.Name = "clamps"


 'OnHand is not a Key property. Its value can be changed.

prod8.OnHand = 22
```

الأنواع المجهولة من تعابير الاستعلام Anonymous Types from Query Expressions

تعابير الاستعلام لا تتطلب دوما إنشاء أنواع مجهولة فعند الإمكان يمكنها استخدام نوع موجود ليحمل بيانات العمود وهذا يحدث عندما يعيد الاستعلام إما سجلات كاملة من مصدر البيانات أو حقل واحد من كل سجل ففي المثال التالي يكون Customers عبارة عن مجموعة فئات Customer والفئة تمتلك العديد من الخصائص بحيث يمكنك تضمين واحدة أو أكثر من هذه الخصائص في نتائج الاستعلام وبأي ترتيب تريده ففي المثالين الأوليين لا يوجد حاجة لأي نوع مجهول لأن الاستعلام يجلب عناصر من أنواع معروفة فـ Custs1 يكون من النوع string لأن cust.Name من النوع String و Custs2 هو مجموعة من الأغراض Customers لأن كل عنصر في Customers هو غرض Customer وكامل العنصر تم جلبه بواسطة الاستعلام

```
Dim custs1 = From cust In customers_

            Select cust.Name


Dim custs2 = From cust In customers_

            Select cust
```

ومع ذلك فالأنواع المسماة لا تكون دائما متوفرة حيث يمكنك الاستعلام عن Names و Addresses من أجل هدف معين و ID و Numbers و Location من أجل هدف آخر فهنا تمكنك الأنواع المجهولة من اختيار أية تركيبة من الخصائص وبأي ترتيب بدون أن تضطر في البداية للتصريح عن نوع مسمى جديد ليحمل النتيجة وبدلا عن ذلك يقوم المترجم بإنشاء نوع مجهول لكل تركيبة من الخصائص فمثلا الاستعلام التالي يحدد فقط Name و ID من كل غرض Customer في customers ومن أجل ذلك يقوم بإنشاء نوع مجهول من تلك الخصائص

```
Dim custs3 = From cust In customers_

            Select cust.Name, cust.ID
```

وكل من الاسم والنوع العائدين لخصائص النوع المجهول يتم أخذها من محددات الاستعلام cust.Name و Cust.Id وتكون خصائص النوع المجهول التي ينشئها الاستعلام خصائص مفتاحية دوما وعند تنفيذ cust3 في حلقة For...Each التالية تكون النتيجة هي مجموعة أنواع مجهولة تمتلك خاصيتين مفتاحيتين Name و ID

```
For Each selectedCust In custs3

  Console.WriteLine(selectedCust.ID & ": " & selectedCust.Name(

Next
```

تحديد متى نستخدم الأنواع المجهولة

قبل أن تقوم بالتصريح عن غرض بأنه من نوع مجهول يجب عليك التفكير فيما إذا كان هذا الخيار هو الأفضل فمثلا إن كنت تريد إنشاء غرض مؤقت ليحتوي بعض حقول المعلومات ولست بحاجة إلى بقية الحقول والطرائق التي تحتويها الفئة الكاملة يكون عندها النوع المجهول حلا جيدا وتكون الأنواع المجهولة ملائمة عندما تريد انتقاء مجموعة مختلفة من الخصائص عند كل تصريح أو إن كنت تريد تغيير ترتيب هذه الخصائص وإن كان مشروعك يحتوي على عدة أغراض تحمل نفس الخصائص بترتيب ثابت يمكنك عندها التصريح عنهم بسهولة باستخدام الأنواع المسماة باستخدام باني فئة فعندها باستخدام باني ملائم يمكن تعريف عدة متغيرات من الفئة Product ويكون ذلك أسهل من استخدام عدة متغيرات مجهولة النوع

```
'Declaring instances of a named type.

Dim firstProd1 As New Product("paperclips", 1.29)

Dim secondProd1 As New Product("desklamp", 28.99)

Dim thirdProd1 As New Product("stapler", 5.09)

'Declaring instances of an anonymous type.

Dim firstProd2 = New With {Key .Name = "paperclips", Key .Price = 1.29}

Dim secondProd2 = New With {Key .Name = "desklamp", Key .Price = 28.99}

Dim thirdProd2 = New With {Key .Name = "stapler", Key .Price = 5.09}
```

وتكمن فائدة أخرى للأنواع المجهولة في أن المترجم يمكنه التقاط الأخطاء الطباعية في أسماء الخصائص ففي المثال السابق يفترض بالأنواع firstProd2 و secondProd2 و thirdProd2 أن تكون متغيرات لنفس النوع المجهول ومع ذلك قمت عن طريق الخطأ بالتصريح عن thirdProd2 بأحد الطرائق اللاحقة وهو نوع مختلف عن firstProd2 و secondProd2

```
'Dim thirdProd2 = New With {Key .Nmae = "stapler", Key .Price = 5.09}

'Dim thirdProd2 = New With {Key .Name = "stapler", Key .Price = "5.09"}

'Dim thirdProd2 = New With {Key .Name = "stapler", .Price = 5.09}
```

والأمر الأهم هو أنه هناك حدود لاستخدام الأنواع المجهولة لا تنطبق على الأنواع المعروفة فمع أن firstProd2 و secondProd2 و thirdProd2 هي متغيرات لنفس النوع المجهول فالمتغير المجهول المشترك غير متوفر ولا يمكن توقع ظهوره كنوع معروف في الكود

فمثلا يمكن استخدام النوع المجهول لتحديد توقيع الطريقة للتصريح عن حقل متغير فيكون بالنتيجة النوع المجهول غير ملائم لتبادل البيانات عبر الطرائق

التحكم PropertyGrid

يقوم التحكم PropertyGrid بعرض معلومات حول الكائن Object بأسلوب مشابه لأسلوب نافذة الخصائص Properties Window في بيئة التطوير وهو يمكن المستخدم من تنظيم الخصائص بحسب الفئة أو أبجديا ممكنا إياه من تحرير قيم هذه الخصائص.
ويمتلك هذا التحكم العديد من العناصر ولكن الأهم من بين هذه العناصر هي الخاصية SelectedObject التي تستخدم لضبط أو معرفة الكائنات المرتبطة مع التحكم وتكون قيمتها من النوع Object كما في الكود

```
' Show Button1 Properites
Me.PropertyGrid1.SelectedObject = Button1

' Show Cla Properties
Dim cla As New Class1
Me.PropertyGrid1.SelectedObject = cla
```

والتحكم PropertyGrid يقوم بعرض خصائص التحكم أو الفئة Properties فقط فهو لا يعرض المتغيرات العامة Public Variables مثلا وهو يعرض من هذه الخصائص تلك القابلة للاستعراض فأي خاصية معلمة بالواصفة Browsable(False) لن يتم عرضها كما تستخدم الواصفة Category لتنظيم خصائص التحكم ضمن أقسام والواصفة Description لإظهار وصف لتلك الخاصية حيث يتم ضبط قيم هذه الواصفات بالشكل المناسب ولكل خاصية على حدى عند كتابة كود الفئة أو التحكم

فإذا افترضنا أنه لدينا فئة باسم Class1 نريد عرض خصائصها في التحكم PropertyGrid وكان كود هذه الفئة كما يلي

```
Imports System.ComponentModel

Public Class Class1
    Public Enum ProgrammingLanguageEnum
        VisualBasic
        CSharp
        CPlusPlus
        Java
        Pascal
    End Enum

    Private _mTest As String
    Private _mName As String
    Private _mProg As ProgrammingLanguageEnum

    <Browsable(False)> _
    Public Property Test() As String
        Get
            Return _mTest
        End Get
        Set(ByVal value As String)
            _mTest = value
        End Set
    End Property

    <Category("Personal"), Description("Person Name.")> _
    Public Property Name() As String
        Get
            Return _mName
        End Get
        Set(ByVal value As String)
            _mName = value
        End Set
    End Property

    <Category("Lang"), Description("His Programming Language")> _
```

```
    Public Property ProgrammingLanguage() As ProgrammingLanguageEnum
        Get
            Return _mProg
        End Get
        Set(ByVal value As ProgrammingLanguageEnum)
            _mProg = value
        End Set
    End Property

End Class
```

يمكننا كتابة الكود التالي لإظهار خصائص المتغير cla الذي هو عبارة عن تواجد ما لتلك الفئة كما يلي

```
Dim cla As New Class1
Me.PropertyGrid1.SelectedObject = cla
```

ففي هذه الفئة استخدمنا في البداية استيرادا لمجال الأسماء System.ComponentModel لكي نتمكن من ضبط بعض الواصفات الخاصة بخصائص فئتنا والتي تؤثر على طريقة إظهار هذه الخصائص في التحكم PropertyGrid حث تلاحظ أن الخاصية Test مزينة بالواصفة Browsable مضبوطة قيمتها إلى False لذا فهي ستسبب عدم ظهور الخاصية Test في التحكم PropertyGrid عندما يعرض خصائص الفئة Class1 ويمكن استخدام الواصفة Category لضبط القسم الذي نرغب في إظهار الخاصية فيه ضمن التحكم PropertyGrid فمثلا تم ضبط الواصفة Category إلى Personal بالنسبة للخاصية Name التي ستلاحظ ظهورها ضمن قسم Personal عند عرض خصائص الفئة والواصفة Description تمكنك من كتابة وصف للخاصية يظهر أسفل نافذة التحكم PropertyGrid

التحويل بين أنواع البيانات باستخدام التضييق Explicit والتوسيع Implicit

لنفرض أنه لدينا برنامج من النوع كونسول مسمى TypeConversions كما هو ظاهر في الكود التالي

```
Module Program

    Sub Main()
        Console.WriteLine("***** The Amazing Addition Program *****")
        Dim a As Short = 9
        Dim b As Short = 10
        Console.WriteLine("a + b = {0}", Add(a, b))
    End Sub

    Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
        Return x + y
    End Function

End Module
```

لاحظ أن الطريقة Add تتوقع أن يمرر لها محددين من النوع Integer ومع ذلك نلاحظ أن الإجراء Main يمرر لها في الواقع محددين من النوع Short وقد يبدو هذا عدم تطابق كلي في أنواع البيانات إلا أن البرنامج يتم تنفيذه وترجمته بدون أخطاء ويعيد القيمة 19 كما هو متوقع. ويعود السبب في ذلك إلى أنه لا يوجد احتمال لحدوث فقدان في البيانات بما أن القيمة الأعظمية للنوع Short هي 32767 بينما تكون للنوع Integer هو 217483647 فيقوم المترجم آليا بتوسيع كل قيمة من النوع Short إلى النوع Integer وذلك بعملية نسخ تلقائية من Short إلى Integer. وبالرغم من أن عملية التعريض الآلية قد تمت لصالحنا في المثال السابق ولكن في بعض الحالات قد تتسبب في أخطاء زمن التنفيذ يصعب كشفها.

افترض أنك قمت بتعديل قيم المتغيران في المثال السابق بحيث عندما يتم جمعهما معا يتسببان في حدوث فيضان على القيمة الأعظمية للنوع Short وبافتراض أنك تقوم بتخزين القيمة المعادة من الطريقة Add ضمن متغير محلي من النوع Short إضافة إلى إظهار النتيجة على الكونسول

```
Module Program

    Sub Main()
        Console.WriteLine("***** The Amazing Addition Program *****")
        Dim a As Short = 30000
        Dim b As Short = 30000
        Dim answer As Short = Add(a, b)
        Console.WriteLine("a + b = {0}", answer)
    End Sub

    Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
        Return x + y
    End Function

End Module
```

فعلى الرغم من أن البرنامج تتم ترجمته بصورة صحيحة هنا إلا أنه عندما يتم تنفيذ البرنامج فسوف يتم إطلاق خطأ زمن التنفيذ System.OverflowException وبالرغم من أن الطريقة Add يمكنها إعادة Integer يحمل القيمة الصحيحة التي تقع ضمن مجال قيم النوع Integer ولكن لا يمكن تخزين تلك القيمة في متغير من النوع Short لأن القيمة تقع خارج مجال القيم التي يمكن تخزينها ضمن النوع Short ففي هذه الحالة يحاول المترجم القيام بعملية تضييق التي ستنتج خطأ فيضان في زمن التنفيذ. ولكن ليست جميع عمليات التضييق ينتج عنها الخطأ System.OverflowwException انظر للمثال التالي

```
Dim myByte As Byte
Dim myInt As Integer = 200
myByte = myInt
Console.WriteLine("Value of myByte: {0}", myByte)
```

ففي الكود السابق تكون القيمة المحتواة في المتغير Integer تقع ضمن مجال النوع Byte لذا فتحويل التضييق لا يتسبب في إطلاق خطأ زمن التنفيذ ومع أن العديد من تحويلات التضييق تعتبر آمنة فإنه من الأفضل تعقب تحويلات التضييق عند الترجمة قبل أن نصل إلى زمن التنفيذ ويتم ذلك باستخدام توجيه المترجم Option Strict

فهم Option Strict

يتأكد Option Strict في زمن الترجمة عوضا عن زمن التنفيذ من تحويلات التضييق ويعطينا إعلاما بحيث يمكننا تصحيحه في وقت مبكر فعندما نستطيع تحديد تحويلات التضييق هذه يمكننا اتخاذ إجراءات تصحيحية وتخفيف خطر ظهور أخطاء في زمن التنفيذ. ومع أنه يمكنك تحديد هذا الخيار في بداية كل ملف كود

```
Option Strict On
```

إلا أنه يمكننا أيضا تفعيل التوجيه Option Strict أو إيقافه للمشروع ككل وذلك بتحديد الخيار المناسب من صفحة Compile من خصائص MyProject فعندما تقوم بتفعيل هذا الخيار فأنت تعلم المترجم بأن عليه التحقق من هذه الاحتمالات خلال عملية الترجمة وسوف يولد لك خطأ في زمن الترجمة عن كل عملية تحويل تضييق.

ويمكننا إضافة الخيار Option Strict لمثالنا السابق ليقوم المترجم بالتحقق من عمليات التضييق حيث يمكننا تغيير نوع المتغير الذي سيحمل النتيجة من النوع Short إلى النوع Integer الذي يمكنه حمل النتيجة بأمان

```
Dim answer As Integer = Add(a, b)
```

دالات التحويل المحدد

يزودنا فيجول بايزيك بعدد من دالات التحويل التي تمكننا من التحويل بين الأنواع بأمان مثل CBool و CByte و CDate و CDbl و CInt إضافة إلى الوظيفة CType التي تأخذ محددين الأول هو البيانات التي لديك والثاني هو نوع البيانات المراد فمثلا سطري الكود التاليين يعتبران متكافئين

```
myByte = CByte(myInt)
myByte = CType(myInt, Byte)
```

وتكمن الفائدة في الوظيفة CType هي أنها تستطيع التعامل مع جميع عمليات التحويل بين مختلف الأنواع والفئات الأساسية والمشتقة وكذلك الأغراض وواجهاتها

دور System.Convert

من أجل إكمال الحديث عن موضوع التحويل يجب علينا إلقاء نظرة على مجال الأسماء System الذي يحتوي على الفئة Convert التي يمكن استخدامها أيضا في عمليات تحويل التضييق والتعريض كما في المثال

```
myByte = Convert.ToByte(myInt)
```

وتكمن الفائدة من استخدام الفئة System.Convert في أنها توفر طريقة لغوية طبيعية للتحويل بين مختلف أنواع البيانات ومع ذلك يزودنا فيجول بايزيك 2008 بكم جيد من وظائف التحويل المضمنة مثل CBool و CByte حيث أن استخدام الفئة Convert للقيام بعمليات التحويل لا يعدو كونه خيار تفضيل شخصي

الفئة StringBuilder

تخزن الفئة StringBuilder سلاسل نصية ديناميكية وتقدم لنا طرائق للتعامل معها وهي تعتبر أسرع بكثير من الفئة String ولكنها تستخدم قدرا من الذاكرة أكبر بشكل ملحوظ من تلك التي تستخدمها الفئة String وحتى تستطيع استخدام الفئة StringBuilder في مشروعك عليك إضافة مرجع لمجال الأسماء System.Text وبافتراض أنك قمت باستيراد مجال الأسماء المذكور يمكننا تعريف متغير يشير إلى تلك الفئة

```
Dim txt As New StringBuilder
```

وبما أنها تتعامل مع سلاسل نصية ديناميكية فيعتبر من الجيد تعريف حجم النص الذي ستقوم بتخزينه فيها مسبقا وتكون السعة الافتراضية 16 محرفا وهذه السعة تتضاعف تلقائيا في كل مرة تقوم بتجاوزها ولضبط هذه السعة نقوم بضبط قيمة الخاصيةCapacity.

ولإنشاء كيان جديد من الفئة StringBuilder يمكنك استدعاء باني الفئة بدون تمرير أية محددات أو تمرير قيمة نصية افتراضية

```
Dim txt As New StringBuilder("some string")
```

وإن كنت قادرا على تقدير طول النص الذي ستقوم بتخزينه يمكنك تمرير هذه القيمة لباني الفئة وبهذا لم يعد هناك تغييرات مستمرة في السعة عند تخزين المحارف في StringBuilder

```
Dim txt As New StringBuilder(initialCapacity)
```

ومع ذلك فالسعة التي قمت بتحديدها ليست نهائية إذ يمكن أن تتغير أثناء التنفيذ وستقوم الفئة StringBuilder بضبط قيمة السعة آليا وإن كنت ترغب بتحديد سعة عظمى لما سيتم تخزينه يمكنك استخدام الباني التالي

```
Dim txt As New StringBuilder( initialCapacity, maxCapacity)
```

وعندما تريد إنشاء كيان جديد من الفئة StringBuilder مستخدما كلا من السعة الابتدائية والسعة العظمى والقيمة الابتدائية يمكنك استخدام باني الفئة بالشكل التالي

```
Dim txt As New StringBuilder( string, initialCapacity, maxCapacity)
```

رأينا حتى الآن خاصيتان أساسيتان للفئة StringBuilder هما Capacity و MaxCapacity وإضافة لهما تقدم لنا هذه الفئة الخصائص Length و Chars والتي تماثل الخصائص التي تحمل نفس الاسم في الفئة String فالخاصية Length تعيد عدد المحارف في الكيان الحالي للفئة StringBuilder والخاصية Chars تعيد مصفوفة محارف وعكس الخاصية Chars الخاصة بالفئة String فهذه الخاصية قابلة للقراءة والكتابة ويكون ترتيب العنصر الأول في هذه المصفوفة هو الصفر وتمتلك الفئة StringBuilder العديد من الطرائق المماثلة لتلك التي تحمل نفس الاسم في الفئة String ولكنها تتعامل مباشرة مع النص الذي يتم تطبيقها عليه ولاتعيد سلسلة نصية جديدة

Append

تضيف الطريقة Append نوعا أساسيا للكيان الحالي للفئة StringBuilder وتكون صيغتها على الشكل التالي حيث يمكن أن يكون المحدد value محرف مفرد أو نص أو تاريخ أو أية قيمة رقمية

```
SB.Append(value)
```

فعندما تقوم بإضافة قيمة رقمية إلى StringBuilder يتم تحويلها إلى نص ثم يتم إضافتها إلى النص المخزن في StringBuilder كما يمكن إضافة قيمة من النوع Object إلى StringBuilder حيث تكون القيمة الفعلية المضافة هي القيمة المعادة من الخاصية ToString العائدة لـ Object كما يمكنك إضافة مصفوفة محارف مستخدمين الصيغة التالية

```
SB.Append(chars, startIndex, count)
```

أو يمكنك إضافة قسم من نص بتحديد مكان بدء ذلك القسم وعدد المحارف المراد إضافتها

```
SB.Append(string, startIndex, count)
```

AppendFormat

الطريقة AppendFormat مشابهة للطريقة Append إلا أنها تقوم بتنسيق النص قبل إضافته ويكون للنص المضاف تحديد معين للتنسيق إضافة إلى قيمته وتكون الصيغة العامة للطريقة AppendFormat كما يلي

```
SB.AppendFormat(string, values)
```

والمحدد الأول يكون نص يحدد خصائص التنسيق و values تكون عبارة عن مصفوفة من القيم واحدة من أجل كل تحديد تنسيق معين وفي المحدد string إذا كان لديك عدد قليل من القيم لتنسيقها وحتى أربعة قيم يمكنك تمريرها كمحددات منفصلة مفصولة بفاصلة

```
SB.AppendFormat(string, value1, value2, value3, vlaue4)
```

والكود التالي يضيف النص Your balance as of Thursday, August 2, 2007 is $19,950.40 للمتغير StringBuilder

```
Dim statement As New StringBuilder
statement.AppendFormat( _
        "Your balance as of {0:D} is ${1: #,###.00}", _
        #8/2/2007#, 19950.40)
```

وكل تحديد لصيغة يتم حصره بقوسين من الشكل {} ويتم ترقيمهم بالتسلسل ابتداء من الصفر ثم تتبع فاصلتهم بالتحديد الفعلي والتنسيق D يخبر الطريقة AppendFormat للقيام بتنسيق النص المحدد بتنسيق التاريخ الطويل والتنسيق الثاني يستخدم فاصل الآلاف وعددان عشريان بعد الفاصلة العشرية من أجل القيمة حيث يضيف الكود التالي نفس النص ولكنها تمرر القيم عبر مصفوفات

```
Dim statement As New StringBuilder
Dim values() as Object = {#8/2/2007#, 19950.4}
statement.AppendFormat( _
       "Your balance as of {0:D} is ${1:#,###.00} ", values)
```

وفي كلتا الحالتين سيحمل المتغير statement نصا كما يلي

Your Balance as of Thursday, August 2, 2007 is $19,950.40

Insert

تضيف هذه الطريقة نصا للكيان الحالي للفئة StringBuilder وتكون صيغتها

SB.Insert(index , value)

حيث أن المحدد index يحدد المكان الذي سيتم إدراج النص الجديد فيه و value هو النص الذي سيتم إدراجه وكما في الطريقة Append يمكن أن تكون value من النوع Object وتكون بالتالي القيمة المدرجة هي القيمة المعادة من الطريقة ToString العائدة لـ Object الأمر الذي يعني أنه يمكنك استخدام الطريقة Insert لإدراج قيم عددية وتواريخ مباشرة ضمن المتغير من النوع StringBuilder وتمكننا صيغة معدلة قليلا من الطريقة Insert من إدراج عدة نسخ من نص معين ضمن StringBuilder

SB.Insert(index, string, count)

كما يمكن استخدام Insert لإدراج مصفوفة محارف في موقع محدد بالمحدد index في الكيان الحالي لـ StringBuilder حيث أن المحدد chars هو عبارة عن مصفوفة محارف

SB.Insert(index, chars)

Remove

هذه الطريقة تزيل عددا من المحارف من StringBuilder الحالي بدءا من موقع محدد وتكون صيغتها كالتالي حيث أن المحدد startIndex هو موقع أول محرف ستتم إزالته من النص والمحدد count هو عدد المحارف التي سيتم إزالتها

SB.Remove(startIndex, count)

Replace

هذه الطريقة تقوم باستبدال جميع تواجدات نص ضمن StringBuilder بنص آخر وتكون صيغتها كالتالي حيث يمكن أن تكون قيمة أي من المحددين نصا أو محرفا

SB.Replace(oldValue, newValue)

وعكس ما يحصل في الفئة String فعملية الاستبدال تتم في الكيان الحالي للفئة StringBuilder وهذه الطريقة لا تعيد نصا آخر كما يمكننا استخدام شكل آخر من الطريقة Replace لتحديد الاستبدالات في قطعة محددة من كيان StringBuilder

SB.Replace(oldValue, newVlaue, startIndex, count)

هذه الطريقة تستبدل جميع تواجدات oldValue بقيمة newVlaue في القسم المحدد وبدايته من startIndex ويمتد عددا من المحارف بقيمة count ابتداء من ذلك الموقع الابتدائي

ToString

تستخدم هذه الطريقة لتحويل كيان StringBuilder إلى String وضبط القيمة المعادة في متغير من النوع String وهي تعيد نصا يمثل القيمة المحتواة في المتغير من النوع StringBuilder التي تم تطبيقها عليها

الواصفة Obsolete Attribute

إليكم   تقديما بسيطا لواصفة تسمى Obsolete وهي مفيدة عندما تقوم بعمل نسخة مطورة من فئة معينة تستخدمها فقد تكون هناك بعض العناصر - دالة أو وظيفة مثلا - في تلك الفئة تريد وضع عناصر أخرى بديلة عنها هنا يمكنك استخدام الواصفة المذكورة لتعليم ذلك العنصر أنه لم يعد مستخدما وربما تمرر لها قيمة نصية تظهر للمبرمج الذي يستخدم هذه الفئة بأن يستخدم عنصر آخر عوضا عن ذلك العنصر الذي تم تعليمه بواسطة هذه الواصفة

وإذا نظرنا لتعريفها في مكتبة MSDN سنرى

```
Marks the program elements that are no longer in use. This class cannot be
inherited
```

وترجمته للعربية: تقوم بتعليم عنصر في البرنامج بأنه لم يعد مستخدما. وهذه الفئة لا يمكن الوراثة منها

وهي تأتي بإحدى ثلاثة صيغ:

الأولى

```
ObsoleteAttribute()
```

مثال:

```
<ObsoleteAttribute()> Public Function OldFunction() As String
      Return "This is the String from old function".
   End Function 'OldFunction
```

الثانية

```
ObsoleteAttribute(String)
```

حيث يمرر لها محدد عبارة عن رسالة نصية تتضمن حلا بديلا لاستخدام هذا العنصر

مثال

```
<ObsoleteAttribute("This function will be removed from future Versions.Use another
function 'NewFunction'")> _

   Public Function OldFunction() As String
      OldFunction= "This is the String from old function".
   End Function 'OldFunction
```

الثالثة

```
ObsoleteAttribute(String, Boolean)
```

حيث يمرر لها محدد أول يكون عبارة عن رسالة نصية تتضمن حلا بديلا لاستخدام هذا العنصر ويحدد المحدد الثاني ذا النوع البولياني فيما إذا كان يجب اعتبار استخدام هذا العنصر خطأ

مثال

استخدام العنصر لا يعتبر خطأ

```
<ObsoleteAttribute("This function will be removed from future Versions.Use another
function 'NewFunction'", False)> _
```

```
   Public Function OldFunction() As String
      OldFunction= "This is the String from old function".
   End Function 'OldFunction
```

استخدام العنصر يعتبر خطأ

```
<ObsoleteAttribute("This function will be removed from future Versions.Use another
function 'NewFunction'", True)> _

   Public Function OldFunction() As String
      OldFunction= "This is the String from old function".
   End Function 'OldFunction
```

سؤال

هل معنى كلامك انه لم يعد بإمكاننا استخدام الدالة القديمة

الجواب

يمكنك استخدام الدالة القديمة في جميع الحالات إلا عند استخدامك للصيغة الثالثة ووضعت قيمة المحدد الثاني إلى True ولكن الهدف من هذه الواصفة هو تعليم العنصر أنه أصبح قديم وغير مستخدم وهدف الرسالة في الصيغة الثانية والثالثة توجيه المبرمج مستخدم هذه الفئة لحل بديل عن استخدام هذا العنصر القديم - هذه الواصفة تنطبق على جميع عناصر الفئة وليس الدوال فقط-

هذا مثال إضافي من أجل مزيد من التوضيح
دقق جيدا في المثال التالي الذي يظهر بعضا من حالات استخدم هذه الواصفة.

```
Public Class TestObsolete

    Public Name As String

    ' You Can Use it With Fields Also
    <Obsolete("No Longer Used")> _
    Public FName As String
    Sub New()

    End Sub

    ' Default Obsolete usage
    <Obsolete()> _
    Public Sub Test1()

    End Sub

    ' Passing A Message
    <Obsolete("Not used Anymore 2, Use Test5")> _
    Public Sub Test2()

    End Sub

    ' تمرير رسالة والمحدد الثاني يحدد أنه لا يتم توليد خطأ
    ' نتيجة استخدام هذه الدالة
    <Obsolete("Not used Anymore 3, Use Test5", False)> _
    Public Sub Test3()

    End Sub

    ' تمرير رسالة والمحدد الثاني يحدد أنه سيتم توليد خطأ
    ' نتيجة استخدام هذه الدالة
    <Obsolete("Not used Anymore 4, Use Test5", True)> _
    Public Sub Test4()

    End Sub
```

```
    Public Sub Test5(ByVal abc As String)
        MsgBox(abc)
    End Sub

End Class
```

الآن حاول استخدامها برمجيا. انظر الكود

```
    Private Sub test()
        Dim a As New TestObsolete
        a.Test1()
        a.Test2()
        a.Test3()
        a.Test4()
        a.Test5("test")
    End Sub
```

سترى أن استخدام الدالات Test1 و Test2 و Test3 سيولد تحذيرا Warning عند ترجمة المشروع لدى محاولة تنفيذه وسيتم تنفيذ المشروع بينما استخدام الدالة Test4 سيعطي خطأ Error ولن يتم تنفيذ المشروع

تخزين ملف ما ضمن Exe البرنامج أثناء التطوير واستعادته أثناء التشغيل

في البداية سنحتاج للاستيرادات التالية كي يعمل الكود معنا

```
Imports System.Reflection
Imports System.IO
```

وهذا هو كود الاستعادة مدعما بالتعليقات الكاملة التي تشرح كيفية عمله

```
' إجراء الغرض منه استخراج أي مصدر ثنائي مضمن ضمن الملف التنفيذي للمشروع
' وتخزينه في ملف
Private Sub ExtractBinaryResource(ByVal ToStoreFileName As String, ByVal
EmResourceName As String)
    ' استخدام حلقة اصطياد الأخطاء
    Try
        ' احضار اسم المجمع الحالي
        Dim Ref As Assembly = Assembly.GetExecutingAssembly
        ' انشاء stream يتضمن المصدر الثنائي الموجود في المجمع
        Dim ResStr As Stream = Ref.GetManifestResourceStream(Ref.GetName.Name _
                              & "." & EmResourceName)

        ' إنشاء مصفوفة بايتات لتحمل البيانات الثنائية
        ' وبطول البيانات التي ستتم قراءتها
        Dim TmBuff(ResStr.Length - 1) As Byte
        ' قراءة البيانات الثنائية من الـ Stream
        ' إلى مصفوفة الباتات
        ResStr.Read(TmBuff, 0, ResStr.Length)

        ' إنشاء FileStream يستخدم لتخزين محتويات المصفوفة المؤقتة إلى القرص
        Using Fs As New FileStream(ToStoreFileName, FileMode.Create)
            ' إنشاء كاتب بيانات ثنائية يعتمد على FileStream
            Using Bw As New BinaryWriter(Fs)
                ' كتابة مصفوفة البياتات للقرص
                Bw.Write(TmBuff)
                ' كتابة جميع البيانات للقرص وتفريغ مخازن الذاكرة المؤقتة
                Bw.Flush()
            End Using
        End Using

    Catch ex As Exception
        ' إظهار رسالة بالخطأ في حال حدوثه
        MsgBox(ex.Message)
    End Try
End Sub
```

والإجراءات الصحيحة لتخزين الملف أثناء التطوير حتى لانواجه مشاكل أثناء التنفيذ

١. انسخ الملف الذي تريد استخدامه لمجلد المشروع مباشرة إياك ونسخه لداخل المجلد Bin كما يفعل الكثير – ولا تضف أي ملفات Resourceلا داعي لها
٢. في الـ Solution Explorer انقر بزر الفأرة اليميني على المشروع ثم اختر Add/Existing Item وأضف الملف الذي تريده للمشروع
٣. غير خاصية Build Action للملف المضاف إلى Embedded Resource

ملاحظة: قيمة المتغير EmResourceName حساسة لحالة الأحرف أي عندما تمرر اسم المصدر الثنائي - ملف أكسس مثلا - يجب أن تطابق حالة الأحرف الممررة هنا لحالة الأحرف المكتوب بها الاسم عند تخزين الملف أي تأكد من مطابقة حالة الأحرف بالضبط لاسم الملف كما يظهر بعد إضافته في الخطوة رقم 2

ملاحظة: هذا الكود يعمل مع الكثير من الملفات مثلا ملف MDB أو PDF أو JPG أو BIN أو DAT أو ... استخدم مخيلتك

كما يمكننا تطوير كود الاستعادة بعدة أشكال حسب الحاجة فمثلا يمكننا تعديل الإجراء لينقل الملف إلى MemoryStream عوضا عن Filestream وبهذا يخزن في ذاكرة مؤقتة بحيث يمكن استخدامه ويكون الإجراء المعدل ليعيد memorystream كما يلي:

```
Private Sub ExBinResToMemoryStream(ByVal EmResourceName As String, _
        ByRef RetMemStr As MemoryStream)

    Try
        Dim Ref As Assembly = Assembly.GetExecutingAssembly
        Dim ResStr As Stream = Ref.GetManifestResourceStream(Ref.GetName.Name _
                               & "." & EmResourceName)

        Dim TmBuff(ResStr.Length - 1) As Byte
        ResStr.Read(TmBuff, 0, ResStr.Length)
        RetMemStr = New MemoryStream(ResStr.Length)
        RetMemStr.Write(TmBuff, 0, ResStr.Length)

    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

فإذا افترضنا أن الملف هو صورة باسم Pic0.jpg مثلا وأردنا تحميلها في مربع الصور نستخدم الكود التالي

```
Dim Ms As MemoryStream
ExBinResToMemoryStream("Pic0.jpg", Ms)
Me.PictureBox1.Image = New Bitmap(Ms)
Ms.Close()
Ms.Dispose()
```

وهذا مثال عن استخدام الكود الأول الذي يستعيد إلى ملف على القرص

```
' فتح صندوق حوار فتح
If Me.SFD.ShowDialog = Windows.Forms.DialogResult.OK Then
    ' الثنائي استدعاء الإجراء وتمرير اسم الملف واسم المصدر
    ExtractBinaryResource(Me.SFD.FileName, "test.mdb")
    ' عرض رسالة بعد الانتهاء
    MsgBox("Ok")
End If
```

تشفير الأسرار للمستخدم الحالي

يحتاج التطبيق في كثير من الأحيان إلى تخزين بيانات خصوصية في ملف أو في الذاكرة والحل الواضح هو التشفير المتماثل الذي يشفر البيانات باستخدام سلسة عشوائية من البايتات تدعى بالمفتاح السري. وتكمن المشكلة عندما تريد فك تشفير البيانات المشفرة في أنك ستحتاج لنفس المفتاح السري الذي استخدمته للتشفير مما يؤدي إلى سلسلة من التعقيدات فإما أنك ستحتاج إلى مكان آمن لتخزين المفتاح السري وهذا أمر صعب أو أنك ستحتاج لاستخلاصه من معلومات أخرى مثل كلمة سر مزودة من قبل المستخدم والتي تكون في الغالب غير آمنة وستنهار كليا عندما ينسى المستخدم كلمة السر الخاصة به.

و الحل المثالي هو جعل نظام الويندوز يقوم بتشفير البيانات من أجلك ولتحقيق هذا ستحتاج إلى ما يسمى بـ DPAPI أو Data Protection API والذي يقوم بتشفير البيانات باستخدام مفتاح متماثل مبني على معلومات خاصة بالمستخدم والآلة وبهذه الطريقة لم تعد تقلق بخصوص تخزين المفتاح ووثوقيته وبدلا عن ذلك يتأكد نظام التشغيل من وثوقية المستخدم عندما يدخل إلى النظام وتكون البيانات المخزنة من قبل مستخدم غير ممكنة الوصول بالنسبة للمستخدمين الآخرين.

في النسخ السابقة من الدوت نيت لايوجد فئات مدارة لاستخدام DPAPI وتم تصحيح هذا في الدوت نيت 2.0 بالفئة الجديدة ProtectData في مجال الأسماء System.Security.Cryptography

كيف يمكنني فعل ذلك

الفئة ProtectData تقدم طريقتان مشتركتان Shared Methods الأولى ProtectData تأخذ مصفوفة بايتات تمثل البيانات المراد تشفيرها وتعيد مصفوفة بايتات بالبيانات المشفرة والثانية UnprotectData تقوم بالعملية المعاكسة حيث تأخذ مصفوفة بايتات بالبيانات المشفرة وتعيد مصفوفة بايتات بالبيانات المفكوك تشفيرها. وباستخدام ProtectData و UnprotectData يمكنك فقط التعامل مع مصفوفات بايتات Byte Array وهذا يعني أنك عندما تريد تشفير بيانات من أي نوع عليك القيام بتحويلها إلى مصفوفة بايتات قبل القيام بالتشفير.

مثال عملي

```
Imports System.Security.Cryptography
Imports System.IO

Module ProtectData

    Sub Main()

        ' Get the data.
        Console.WriteLine("Enter a secret message and press enter.")
        Console.Write(">")
        Dim Input As String = Console.ReadLine()
        Console.WriteLine()

        If Input <> "" Then
            Dim Data(), EncodedData() As Byte

            ' Write the data to a new MemoryStream.
            Dim DataStream As New MemoryStream()
            Dim Writer As New StreamWriter(DataStream)
            Writer.Write(Input)
            Writer.Close()

            ' Convert the MemoryStream into a byte array,
            ' which is what you need to use the ProtectData() method.
            Data = DataStream.ToArray()
```

```
            ' Encrypt the byte array.
            EncodedData = ProtectedData.Protect(Data, Nothing, _
              DataProtectionScope.CurrentUser)

            ' Store the encrypted data in a file.
            My.Computer.FileSystem.WriteAllBytes("c:\secret.bin", _
              EncodedData, False)

        End If
    End Sub

End Module
```

عندما تقوم بتشغيل التطبيق سيطلب من إدخال نص والذي سيقوم بتشفيره باستخدام الحساب الحالي للمستخدم ويقوم بتخزين البيانات في الملف secret.bin والبيانات الموجودة فيه لن يستطيع قراءتها أي مستخدم آخر. وللتأكد من أن البيانات مشفرة فعلا لديك خياران قم بفتح الملف وألقي نظرة بنفسك أو يمكنك تعديل الكود ليقرأ البيانات مباشرة من مجرى الذاكرة Memory Stream و الكود التالي يحاول القيام بالخيار الثاني ونتيجة إظهاره ستتكون سلسة نصية لا معنى لها

```
' Verify the data is encrypted by reading and displaying it
' without performing any decryption.
DataStream = New MemoryStream(EncodedData)
Dim Reader As New StreamReader(DataStream)
Console.WriteLine("Encrypted data: " & Reader.ReadToEnd())
Reader.Close()
```

ولتفكيك تشفير البيانات يجب عليك وضعها في مصفوفة بايتات ثم استخدام الطريقة UnprotectData لاستخلاص البيانات من مصفوفة البايتات ويمكنك إضافة StreamReader لإضافة دعم لتفكيك التشفير للمثال السابق مثلا يمكنك إضافة الكود التالي ليقرأ البيانات من الملف ويظهر الجملة التي أدخلتها سابقا

```
If My.Computer.FileSystem.FileExists("c:\secret.bin") Then
     Dim Data(), EncodedData() As Byte

     EncodedData = My.Computer.FileSystem.ReadAllBytes("c:\secret.bin")
     Data = ProtectedData.Unprotect(EncodedData, Nothing, _
     DataProtectionScope.CurrentUser)

     Dim DataStream As New MemoryStream(Data)
     Dim Reader As New StreamReader(DataStream)

     Console.WriteLine("Decoded data from file: " & Reader.ReadToEnd())
     Reader.Close()
     Console.WriteLine()
End If
```

تذكر بما أن البيانات تم تشفيرها بواسطة حساب المستخدم الحالي يمكنك تفكيك تشفيرها في أي وقت والقيد الوحيد هو انك يجب أن تدخل باستخدام نفس حساب المستخدم ولاحظ أنك عندما تقوم بحماية البيانات يجب عليك اختيار واحدة من القيم من التعداد DataProtectionScope Enumeration ويكون بذلك لديك خياران:

LocalMachine

سيقوم ويندوز بتشفير البيانات بمفتاح خاص بالآلة وبهذا تضمن أن لا أحد يستطيع قراءة البيانات إلا على نفس الجهاز. وهذا يعمل جيدا بالنسبة للتطبيقات التي تعمل على المخدم Server Side Application والتي تعمل بدون تدخل المستخدم

CurrentUser

سيقوم ويندوز بتشفير البيانات بمفتاح خاص بالمستخدم وبالتالي لا يمكن قراءتها من قبل المستخدمين الآخرين

في المثال المذكور يتم تخزين بيانات خاصة بالمستخدم ومع ذلك يمكنك تغيير مجال حماية البيانات DataProtectionScope ليتم تخزين البيانات بشكل يستطيع جميع مستخدمي الجهاز الوصول إليها.

وفيما يلي سرد للمثال كاملا وهو تطبيق من نوع Console Application

```
Imports System.Security.Cryptography
Imports System.IO

Module ProtectData
    Sub Main()

        ' Get the data.
        Console.WriteLine("Enter a secret message and press enter.")
        Console.Write(">")
        Dim Input As String = Console.ReadLine()
        Console.WriteLine()

        If Input <> "" Then
            Dim Data(), EncodedData() As Byte

            ' Write the data to a new MemoryStream.
            Dim DataStream As New MemoryStream()
            Dim Writer As New StreamWriter(DataStream)
            Writer.Write(Input)
            Writer.Close()

            ' Convert the MemoryStream into a byte array,
            ' which is what you need to use the ProtectData() method.
            Data = DataStream.ToArray()

            ' Encrypt the byte array.
            EncodedData = ProtectedData.Protect(Data, Nothing, _
              DataProtectionScope.CurrentUser)

            ' Store the encrypted data in a file.
            My.Computer.FileSystem.WriteAllBytes("c:\secret.bin", _
               EncodedData, False)

            ' Verify the data is encrypted by reading and displaying it
            ' without performing any decryption.
            DataStream = New MemoryStream(EncodedData)
            Dim Reader As New StreamReader(DataStream)
            Console.WriteLine("Encrypted data: " & Reader.ReadToEnd())
            Reader.Close()
            Console.WriteLine()
        End If

        If My.Computer.FileSystem.FileExists("c:\secret.bin") Then
            Dim Data(), EncodedData() As Byte

            EncodedData = My.Computer.FileSystem.ReadAllBytes("c:\secret.bin")
            Data = ProtectedData.Unprotect(EncodedData, Nothing, _
              DataProtectionScope.CurrentUser)

            Dim DataStream As New MemoryStream(Data)
            Dim Reader As New StreamReader(DataStream)
```

```
            Console.WriteLine("Decoded data from file: " & Reader.ReadToEnd())
            Reader.Close()
            Console.WriteLine()
        End If

        Console.ReadLine()
    End Sub

End Module
```

ماذا عن ...

حماية البيانات قبل تخزينها في قاعدة البيانات؟ طالما أنك استخدمت الفئة ProtectedData لتشفير بياناتك يمكنك وضعها في أي مكان تريده ففي المثال السابق قمت بكتابة البيانات المشفرة إلى ملف ومع ذلك يمكنك كتابة البيانات الثنائية إلى سجل قاعدة البيانات ولفعل ذلك ستحتاج ببساطة إلى حقل ثنائي binary في جدولك بمساحة كافية ليتسع لمصفوفة البايتات المشفرة وفي الـ Sql Server تستخدم نوع البيانات varbinary لهذا الغرض

توجيهات المعالج

الترجمة الشرطية

يمكننا استخدام الترجمة الشرطية لاختيار مقاطع معينة من الكود لترجمتها فمثلا ربما ترغب في كتابة عبارات للتنقيح لمقارنة السرعة عند استخدام طرق مختلفة لتحقيق مهمة برمجية معينة أو ربما ترغب في كتابة كود معين من اجل منطقة لغوية معينة وتم تصميم عبارات الترجمة الشرطية ليتم تنفيذه أثناء عملية الترجمة.

يمكنك التصريح عن ثابت للترجمة الشرطية باستخدام التوجيه #Const ويمكنك بعدها تحديد مقاطع الكود المراد ترجمته شرطيا باستخدام بلوك #If ... Then ... #Else ... #ElseIf ....#End If فمثلا يمكننا التصريح عن ثابت نستخدمه لتحديد هل نقوم بترجمة نسخة كاملة أو نسخة للعرض فقط مستخدمين الثابت #Const كما في المثال

```
#Const CompDemo = "Demo"

#If CompDemo = "Demo" Then
    ' العرض بنسخة الخاص الكود
#ElseIf CompDemo = "Special" Then
    ' الخاصة بالنسخة الخاص الكود
#Else
    ' العادية بالنسخة الخاص الكود
#End If
```

بالإضافة إلى إمكانية تعريفك للثوابت التي تحتاجها بناء على احتياجات برنامجك هناك ثوابت تأتي معرفة سابقا وهي

- **CONFIG** نص يعطينا الضبط الحالي لتعريف الحل المفعل حاليا Active Solution Configuration في صندوق Configuration manager
- **DEBUG** قيمة منطقية يمكن ضبطها من صندوق حوار خيارات المشروع وهي تحدد هل ترجمة البرنامج حاليا موضوعة على وضع التنقيح في الحالة الافتراضية
- **TARGET** نص يحدد نوع خرج المشروع والقيم الممكنة لهذا الثابت هي winexe أو exe أو library أو module
- **TRACE** قيمة منطقية يمكن تحديدها من خصائص المشروع
- **VBC_VER** تحتوي على رقم نسخة الفيجول بايزيك
- **_MYTYPE** سلسلة نصية تمثل نوع المشروع الذي يتم بناؤه وهذه تتحكم في أي من أغراض My تتوفر في المشروع

طوي وإخفاء أقسام من الكود

تستخدم #Region من أجل طوي وإخفاء أقسام من الكود وهي تستخدم من أجل تنظيم الكود وتصنيفه ضمن أقسام وهي تأخذ محدد وحيد هو سلسلة نصية تحدد اسما للقسم كما في المثال

```
#Region " General Functions "
    ' القسم ضمن الكود
#End Region
```

#ExternalSource

وهو يستخدم حصريا من قبل المترجم والمنقح حيث يوفر ربط بين سطور محددة من الكود المصدري ونص خارجي وصيغته العامة

```
#ExternalSource( StringLiteral , IntLiteral )
    [ LogicalLine+ ]
#End ExternalSource
```

كيف تجعل لنافذة برنامج ظلا

استخدم الكود التالي لتجاوز الخاصية CreateParams للنافذة وإضافية خاصية الظل لها

```
Public Class Form1
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code

    Protected Overrides ReadOnly Property CreateParams() _
                        As System.Windows.Forms.CreateParams

        Get
            Const CS_DROPSHADOW = &H20000
            Dim cp As CreateParams = MyBase.CreateParams
            cp.ClassStyle = cp.ClassStyle Or CS_DROPSHADOW
            Return cp
        End Get
    End Property

End Class
```

كيف تقوم بعمل أيقونة خاصة لتحكمك الخاص

عندما تقوم بإنشاء تحكم خاص بك فيجب أن تكون له أيقونة تميزه ولاستبدال الأيقونة الافتراضية التي يظهرها الفيجول ستوديو في شريط الأدوات نستخدم ملف bmp أو ملف ico مستقلين أو مضمنين ضمن الـ dll الناتج عن المشروع ولكن يجب عليك الانتباه إلى أن لون البكسل اليساري الأسفل يحدد اللون الشفاف للأيقونة فإذا كان لون ذلك البكسل أصفر فأينما ورد اللون الأصفر في الأيقونة يصبح شفافا

ولإضافة ملف مضمن للـ DLL استخدم أمر Add new من قائمة Project وأضف ملف الصورة للمشروع وسمه باسم التحكم الذي سيأخذ الأيقونة مثلا إذا كان اسم التحكم لديك TextBoxEx فيجب أن يكون اسم الملف TextBoxEx.bmp أو TextBoxEx.ico ويجب أن تكون أبعاده 16 × 16 بكسل ثم قم باختيار خصائص الملف و غير الخاصية Build Action إلى Embedded Resource ثم أعمل Compile للمشروع وتلاحظ أنك قد نجحت بعمل ذلك دون إضافة إي كود وإذا كان ملف الصورة خارجي عندها ستضطر لاستخدام صفة Attribute ToolboxBitmapفعلى سبيل المثال انظر الكود التالي

```
<ToolboxBitmap("C:\CustomControlDemo\TextBoxEx.ico")> _
Public Class TextBoxEx
§
End Class
```

وهي مفيدة في حالات أخرى مثل أن الصورة موجودة فيDll آخر أو موجودة بنفس الـ Dll و بغير اسم كالمثال

```
<ToolboxBitmap(GetType(TextBoxEx), "TextBoxIcon.bmp")> _
Public Class TextBoxEx
§
End Class
```

حالة أخرى ستحتاج فيها إلى استخدام هذه الصفة Attribute وهي حالة تعشيش الأسماء مثلا التحكم TextBoxEx موجود في مجال الأسماء CustomControlDemo.Controls عندها عليك تغيير اسم الصورة إلى Controls.TextBoxIcon.bmp ومع ذلك ستبقى محتاجا لاستخدام الصفة ToolboxBitmap

لماذا يأخذ كودك وقتا طويلا أثناء التنفيذ

Profiling هو معرفة تصرفات كودك و قسم كبير منه هو معرفة فيما إذا كان هذا الكود يأخذ وقتا طويلا للتنفيذ. كما لا يجب القيام به منذ البدء بالتطوير ولكنه يصبح ضروريا عند تدقيق الأنظمة التي تسير ببطء كما أنه يصبح مفيدا قرب الانتهاء من التطوير وخاصة لتلك الأنظمة التي تعمل خارج المجال المقبول

في فيجول ستوديو 2005 وخاصة إصدارة Team لديها أدوات رائعة لـ Profiling ولكنها مصممة للعمل ضمن بيئة التطوير. و الـ Auto Profiler الذي يبقى مع كودك يجعلك تقرر متى تشغله ومتى توقفه. وهذه المقالة تعرض كيفية توظيف بعض الوظائف المفيدة في الدوت نيت لبناء Auto Profiler سهل الاستخدام يمكنه توقيت سطر واحد أو برنامج كامل

Implementing the Timestamp Class

الخطوة الأولى هي بناء فئة تتعقب وتوقف الأوقات. ويجب أن تعلم أنه عندما تبدأ بتوقيت قطعة من الكود والوقت المار منذ البدء يمكنك استخدام الفئة DateTime من أجل توقيت البداية والنهاية متضمنا حسابات الوقت المنتهي لهذه الفئة والكود التالي يبين فئة Stamp

```
Friend Class Stamp
   Private start As DateTime
   Public Sub New()
      start = DateTime.Now
   End Sub
   Public ReadOnly Property ElapsedTimeString() As String
      Get
         Return ElapsedTime.ToString()
      End Get
   End Property
   Public ReadOnly Property StartTime()
      Get
         Return start.ToLongTimeString()
      End Get
   End Property
   Public ReadOnly Property ElapsedTime() As TimeSpan
      Get
         Return DateTime.Now.Subtract(start)
      End Get
   End Property
End Class
```

Implementing the MarkTime Class

ولإبقاء الفئة سهلة الاستخدام ضع معظم العمل عليك والفئة التالية MarkTime تستخدم تتبع عام لكائن Stamp وهي تبني كائن Stamp وتضعه في الذاكرة وتعيد علامة الوقت وستحتاج هنا للمكدس Stack لتتبع التكرار فمثلا عندما تكرر الطريقة نفسها عشر مرات فإنك تضيف عشر علامات بدء لمكدس MarkTime قبل أن تحتسب أوقات الانتهاء والكود التالي يبين الفئة MarkTime

```
Friend Class MarkTime
   Private stack As Stack(Of Stamp) = Nothing
   Public Sub New()
      stack = New Stack(Of Stamp)()
   End Sub
   Public Function AddStart() As String
      Dim start As Stamp = New Stamp()
      stack.Push(start)
      Return start.StartTime
```

```
    End Function
    Public Function RemoveStart() As String
        If (stack.Peek() Is Nothing = False) Then
            Return stack.Pop().ElapsedTimeString
        Else
            Return ""
        End If
    End Function
End Class
```

بناء AutoProfiler باستخدام جدول التهشيرHashtable

وأخيرا الـ AutoProfiler يحتوي على باني مشترك shared constructor هو Sub New و طريقتان مشتركتان هما Stopp و Start الطريقة Start تستدعي الطريقة المشتركة GetKey التي تستخدم StackTrace والانعكاس Reflection للحصول على الاسم الكامل للطريقة المستدعية وهذا الاسم يصبح المفتاح Key في جدول التهشير Hashtable وهنا المستخدم لن يحتاج لمعرفة أي الطرق يتم قياسها لأن جدول التهشير Hashtable يقوم بذلك وإذا تم استدعاء الطريقة عدة مرات وكان المدخل موجود سابقا في جدول التهشير Hashtable سيتم معاملة التوقفات والابتداءات الإضافية بنفس كائن MarkTime في جدول التهشير Hashtable وكل ما سيحتاجه مستخدم AutoProfilerهو استدعاء AutoProfiler.Start أو AutoProfiler.Stopp وستقوم الفئة Class بمتابعة وقت البدء والتوقف والمستدعي و الكود التالي يحتوي الفئةMarkTime

```
Public Class AutoProfiler
    Private Shared hash As Hashtable = Nothing
    Private Shared output As OutputType = OutputType.Console
    Shared Sub New()
        hash = New Hashtable
    End Sub
    Private Shared Function GetKey() As String
        Const mask As String = "{0}.{1}"
        Dim trace As StackTrace = New StackTrace
        Dim method As MethodBase = trace.GetFrame(2).GetMethod()
        Return String.Format(mask, _
            method.ReflectedType.FullName, method.Name)
    End Function
    Public Shared Property OutputTo() As OutputType
        Get
            Return output
        End Get
        Set(ByVal value As OutputType)
            output = value
        End Set
    End Property
    <Conditional("DEBUG")> _
    Public Shared Sub Start()
        Dim marker As MarkTime = Nothing
        Dim key As String = GetKey()
        If (hash(key) Is Nothing) Then
            marker = New MarkTime()
            hash.Add(key, marker)
        Else
            marker = CType(hash(key), MarkTime)
        End If
        WriteLine("Started {0} at {1}", key, marker.AddStart())
    End Sub
    <Conditional("DEBUG")> _
```

```
    Public Shared Sub Stopp()
        Dim marker As MarkTime = Nothing
        Dim key As String = GetKey()
        If (hash(key) Is Nothing) Then
            Throw New ArgumentOutOfRangeException(key, _
                "Can't find start time entry")
        End If
        marker = CType(hash(key), MarkTime)
        WriteLine("Stopped: {0}, elapsed time {1}", _
            key, marker.RemoveStart())
    End Sub
    Private Shared Sub WriteLine(ByVal format As String, _
        ByVal ParamArray args() As Object)
        If (output = OutputType.Console) Then
            System.Console.WriteLine(String.Format(format, args))
        Else    ' debug
            System.Diagnostics.Debug.WriteLine( _
                String.Format(format, args))
        End If
    End Sub
End Class
```

و الكود التالي يحتوي على النص الكامل لـAutoProfiler متضمنا برنامج Console كمثال يبين سهولة استخدام هذه الطريقة

```
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Diagnostics
Imports System.IO
Imports System.Reflection
Imports System.Text

Module Module1
    Sub Main()
        Test()
    End Sub
    Sub Test()
        Profiler.AutoProfiler.Start()
        System.Threading.Thread.Sleep(5000)
        Profiler.AutoProfiler.Stopp()
        Console.ReadLine()
    End Sub
End Module
Namespace Profiler
    Public Enum OutputType
        Console
        Debug
        Window
    End Enum
    Public Class AutoProfiler
        Private Shared hash As Hashtable = Nothing
        Private Shared output As OutputType = OutputType.Console
        Shared Sub New()
            hash = New Hashtable
        End Sub
```

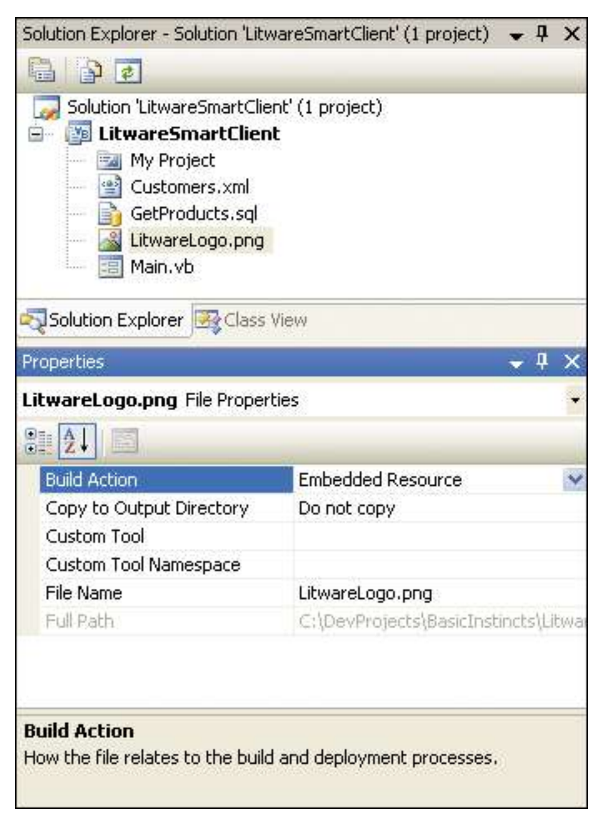
```
   Private Shared Function GetKey() As String
      Const mask As String = "{0}.{1}"
      Dim trace As StackTrace = New StackTrace
      Dim method As MethodBase = trace.GetFrame(2).GetMethod()
      Return String.Format(mask, _
         method.ReflectedType.FullName, method.Name)
   End Function
   Public Shared Property OutputTo() As OutputType
      Get
         Return output
      End Get
         Set(ByVal value As OutputType)
         output = value
      End Set
   End Property
   <Conditional("DEBUG")> _
   Public Shared Sub Start()
      Dim marker As MarkTime = Nothing
      Dim key As String = GetKey()
      If (hash(key) Is Nothing) Then
         marker = New MarkTime()
         hash.Add(key, marker)
      Else
         marker = CType(hash(key), MarkTime)
      End If
      WriteLine("Started {0} at {1}", key, marker.AddStart())
   End Sub
   <Conditional("DEBUG")> _
   Public Shared Sub Stopp()
      Dim marker As MarkTime = Nothing
      Dim key As String = GetKey()
      If (hash(key) Is Nothing) Then
         Throw New ArgumentOutOfRangeException(key, _
            "Can't find start time entry")
      End If
      marker = CType(hash(key), MarkTime)
      WriteLine("Stopped: {0}, elapsed time {1}", _
            key, marker.RemoveStart())
   End Sub
   Private Shared Sub WriteLine(ByVal format As String, _
      ByVal ParamArray args() As Object)
      If (output = OutputType.Console) Then
         System.Console.WriteLine(String.Format(format, args))
      Else ' debug
         System.Diagnostics.Debug.WriteLine( _
            String.Format(format, args))
      End If
   End Sub
End Class
Friend Class MarkTime
   Private stack As Stack(Of Stamp) = Nothing
   Public Sub New()
      stack = New Stack(Of Stamp)()
   End Sub
   Public Function AddStart() As String
```

```
        Dim start As Stamp = New Stamp()
        stack.Push(start)
        Return start.StartTime
      End Function
      Public Function RemoveStart() As String
        If (stack.Peek() Is Nothing = False) Then
          Return stack.Pop().ElapsedTimeString
        Else
          Return ""
        End If
      End Function
    End Class
    Friend Class Stamp
      Private start As DateTime
      Public Sub New()
        start = DateTime.Now
      End Sub
      Public ReadOnly Property ElapsedTimeString() As String
        Get
          Return ElapsedTime.ToString()
        End Get
      End Property
      Public ReadOnly Property StartTime()
        Get
          Return start.ToLongTimeString()
        End Get
      End Property
      Public ReadOnly Property ElapsedTime() As TimeSpan
        Get
          Return DateTime.Now.Subtract(start)
        End Get
      End Property
    End Class
End Namespace
```

توقيت كودك

لقد قمت للتو بإنشاء AutoProfiler يمكن المستخدم من توقيت أي أمر أو عدة أوامر أو حتى قطعات كبيرة من الكود فقط باستدعاء AutoProfiler.Start و AutoProfiler.Stopp حيث تقوم هذه التقنية بتوظيف generics و hashtables و reflection ومعرفة الفئة StackTrace حيث ستجدها مفيدة عندما تواجه كودا يتم تنفيذه بأبطأ من المقبول

ملفات المصادر وتخصيص البرنامج محليا Resources and localization



يوجد طريقتين للاستفادة من المصادر كالنصوص والصور والملفات النصية من داخل تطبيقات الفريموورك حيث يمكنك تضمينهم مباشرة ضمن تطبيقك أو تحميلهم من ملف خارجي، فعندما تختار التحميل من ملف خارجي بدلا من المصادر المضمنة يجب عليك توزيع هذه الملفات مع مجمع التطبيق، كما يجب عليك التأكد من أن الكود داخل التطبيق يستطيع تحديد مسار ملفات المصادر هذه في وقت التنفيذ. وهذا سيسبب مشاكل إذا تم فصل الملف التنفيذي عن هذه الملفات التي يعتمد عليها. فعند أخذ خيار تضمين هذه الملفات ضمن المجمعات التي تستخدمها سيصبح التوزيع عندها أكثر وثوقية وأقل عرضة للأخطاء وسأقوم هنا بشرح استخدام هذه المصادر وكيف ولماذا يجب تضمين هذه المصادر ودور المصادر في الفريموورك

تضمين مصدر

سنبدأ بمثال بسيط لنرى كيف يمكن تحقيق التضمين. افترض أنك تريد تضمين ملف صورة اسمه LitwareLogo.png في تطبيقك ستبدأ بإضافة هذا الملف إلى مشروعك ثم من خلال صفحة الخصائص ستحدد Build Action له إلى Embedded Resource وأنت بعملك هذا قد أرشدت بيئة التطوير إلى تضمين الملف داخل الملف التنفيذي للبرنامج.

بعد قيامك بتضمين الملف كمصدر يجب عليك معرفة كيفية الوصول إليه في زمن التنفيذ. قم بفحص قطعة الكود التالية والتي تحصل على مرجع للمجمع الحالي ثم تستدعي الإجراء GetManifestResourceStream لتحصل على مجرى وصول لملف المصدر -stream based access كما يجب عليك استيراد مجال الأسماء System.Reflection و System.IO لتنفيذ الكود بشكل صحيح

```
'*** get current Assembly object.
Dim asm As Assembly = Assembly.GetExecutingAssembly()
'*** load embedded resource into stream
Dim ResourceName As String = "LitwareSmartClient.LitwareLogo.png"
Dim str As Stream = asm.GetManifestResourceStream(ResourceName)
    '*** convert stream into image and load in     '*** picture box
    Dim img As Image = Image.FromStream(str)
    PictureBox1.Image = img
```

كما ترى هنا فالمجمع يظهر الطريقة GetManifestResourceStream التي تسمح لك بتمرير نص يمثل المصدر المضمن كما يجب عليك الانتباه هنا إلى أن اسم المصدر حساس لحالة الحروف حتى عندما تستخدم لغة برمجة غير حساسة لحالة الحروف مثل فيجول بايزيك ففي هذا المثال الكود يستدعي الطريقة Image.FromStream لتحويل المجرى الذي يحتوي الصورة إلى صورة يمكن تحميلها ضمن صندوق الصورة PictureBox.

لنفترض أنه لديك وثيقة xml كبيرة أو عبارة Sql طويلة أو إجراءات جافا يحتاجها برنامجك يمكنك الإبقاء على هذه الملفات منفصلة داخل مشروعك في هذه الحالة ستستفيد من ميزة تلوين الكود في بيئة التطوير ومخطط xml كل هذا يتطلب تضمين هذه المصادر في المجمع الناتج والوصول إليهم باستخدام هذه الطريقة كما سترى فإذا كان لديك ملف SQL و ملف XML مضمنين داخل برنامجك يمكنك الوصول إليهم كالتالي:

```
Load Resources
'*** get current Assembly object.
Dim asm As Assembly = Assembly.GetExecutingAssembly()

'*** load embedded SQL resources file
Dim SqlResourceName As String = "LitwareSmartClient.GetProducts.sql"
Dim strSQL As Stream = asm.GetManifestResourceStream(SqlResourceName)
Dim reader As New StreamReader(strSQL)
Dim sql As String = reader.ReadToEnd
reader.Close()

'*** load embedded XML resources file
Dim XmlResourceName As String = "LitwareSmartClient.Customers.xml"
Dim strXML As Stream = asm.GetManifestResourceStream(XmlResourceName)
Dim xmlDoc As New XmlDocument()
xmlDoc.Load(strXML)
strXML.Close()
```

ملفات المصادر

التقنية التي رأيتها تتضمن تضمين ملفات المصادر مباشرة داخل المجمع وتحميلهم باستخدام الطريقة GetManifestResourceStream التي تزودها فئة Class المجمع ولكن هناك ملفات مصادر بديلة تسهل التعامل مع المصادر في العديد من الحالات كما أن بيئة التطوير تقدم بعض التسهيلات عندما يتعلق الأمر بالمصادر وتخصيص البرنامج محليا

التعامل مع ملفات المصادر

يمكن استخدام المصادر في الدوت نيت كمصادر مضمنة ضمن المجمعات وإحدى فوائد استخدام ملفات المصادر هي أن جميع اللغات والعناصر المحلية في التطبيق أو المكتبة كالنصوص ورسائل المستخدم يمكن استخراجها من شفرة تطبيقك ولعمل ذلك تقوم بعمل ملف مصادر خاص لكل لغة تريد دعمها وملف المصادر الفعلي هو ملف نصي يعتمد على صيغة xml بلاحقة .resx وهذا مثال يعطيك فكرة عما يمكن أن يحتويه الملف

```
<root>

 <data name="MainFormCaption">
   <value>Litware Customer Manager</value>
 </data>

 <data name="UserWelcome">
   <value>Good day</value>
 </data>

 <data name="ErrorMessage1">
   <value>Oh no, Something went wrong!</value>
 </data>

</root>
```

يمكنك ترجمة ملف المصادر إلى الصيغة الثنائية باستخدام الأداة Resgen.exe وسيكون للملف الناتج اللاحقة .resources كالمثال التالي الذي يمكن تنفيذه من ملف دفعي أو من داخل بيئة التطوير

```
RESGEN.EXE LitwareStrings.resx LitwareStrings.resources
```

وبعدها يجب عليك ترجمة الملف الناتج لمجمع دوت نيت باستخدام الأداة Al.exe كالمثال

```
AL.EXE /t:library  /out:LitwareStrings.resources.dll  /link:LitwareStrings.resources
```

وبعد ترجمة هذه المصادر إلى مجمع دوت نيت عندها يمكنك الوصول إليها باستخدام الفئة ResourceManager الموجودة ضمن مجال الأسماء System.Resources كالمثال

```
Dim asm As Assembly = Assembly.Load("LitwareStrings.resources")
Dim rm As New System.Resources.ResourceManager("LitwareStrings", asm)
Dim caption As String = rm.GetString("MainFormCaption")
```

كما يمكن استخدام Resgen.exe لإنشاء ملفات مصادر بأنواع فئات قوية strongly typed resource class تعرض خصائص تقدم طريقة سهلة للوصول إلى ملفات المصادر كالمثال

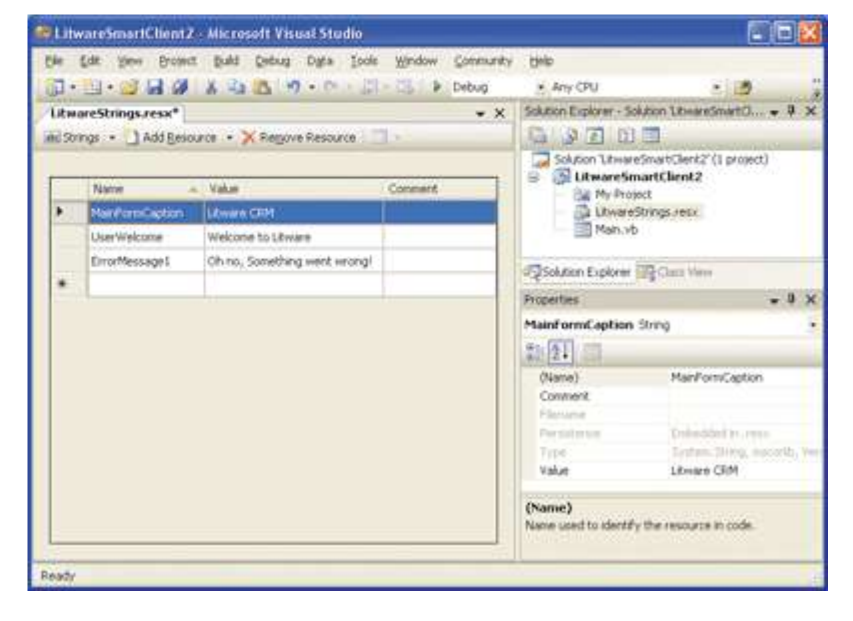
```
RESGEN.EXE LitwareStrings.resx LitwareStrings.resources /str:vb
```

وسطر الأوامر هذا يولد ملف vb يحتوي على فئة تتضمن شفرة تستدعي ResourceManager وتعيد خصائص بالمصادر

```
Shared ReadOnly Property MainFormCaption() As String
  Get
    Return ResourceManager.GetString("MainFormCaption", resourceCulture)
  End Get
End Property
```

لقد انتهيت من عرض سريع بمستوى عالي لكيفية ترجمة ملفات المصادر ضمن المجمعات وكيفية استخدامها من خلال الفئة ResourceManager وملفات المصادر القوية وهذا يعطيك فكرة عن كيفية جمع الملفات المختلفة معا. ولن نكمل الوقت في التفاصيل منخفضة المستوى لأنك لن تضطر للتعامل معها عند تخصيصك للبرنامج محليا في التطبيقات والمكتبات لأن بيئة التطوير تقدم لك تسهيلات كثيرة في هذا المجال. وأبقي في ذهنك أنك ربما ستحتاج للتعامل مع هذه الأوامر عند قيامك ببرمجة بعض الأدوات والمكتبات

ملفات المصادر في فيجول ستوديو 2005



سنركز الآن على استخدام المصادر ضمن برامج نماذج الويندوز Windows Forms-based application ومعظم المفاهيم التي تم عرضها تنطبق أيضا على المكتبات class library DLL ويسهل عليك فيجول ستوديو التعامل مع ملفات المصادر بتقديم محرر مرئي حيث يمكنك إضافة ملف مصادر جديد باستخدام الأمر Add New Item واختيار Resources file وبعد قيامك بإضافة ملف مصادر للمشروع لم يعد من الضروري التعامل معها بالصيغة xml داخل ملف resx. فبدلا عن ذلك يزود ك فيجول ستوديو بمحرر مصادر سهل يمكنك من التعامل مع مختلف أنواع المصادر بسهولة

عندما تترجم compile مشروع يحتوي على ملف مصادر فإن فيجول ستوديو يترجم الملف resx. إلى resources. ثم يربطها داخل الصورة الفيزيائية للمجمع الناتج وهذا يعني أن كافة التفاصيل المتعلقة بترجمة ملفات المصادر وتضمينها ضمن صورة المجمع الناتج يتم توليها من قبل فيجول ستوديو كما أنه يبني فئة مصادر بنوع قوي strongly typed resource class ويكشفها لمشروع فيجول بايزيك ضمن مجال الأسماء My الجديد في 2005 وهذا يعني أنك تحصل على فوائد الفئة ResourceManager class التي تقوم بتحميل مصادرك مباشرة ولم يعد من الضروري قيامك ببرمجة الفئة مباشرة حيث يمكنك استخدام المصادر مباشرة ببساطة كالكود

```
Sub Main_Load(sender As Object, e As EventArgs)
  Handles MyBase.Load
  Me.Text = _
    My.Resources.LitwareStrings.MainFormCaption
  Me.lblUserWelcome.Text = _
    My.Resources.LitwareStrings.UserWelcome
End Sub
```

ملفات المصادر على مستوى المشروع

بينما يمكنك بشكل واضح إضافة ملف مصادر إلى مشروع فيجول ستوديو الأمر الذي يعتبر غير ضروري لوجود ملف مصادر مضمن سلفا على مستوى المشروع يتم إضافته تلقائيا كلما قمت بإنشاء مشروع جديد والذي يمكن الوصول إليه من خصائص المشروع في المحرر وعندما تريد الوصول إلى المصادر المختلفة كالنصوص يمكنك الوصول إليهم مباشرة من خلال الفئة My.Resources

```
Private Sub LoadResources()
  '*** load project-level resources
  Me.Text = My.Resources.MainFormCaption
  Me.lblWelcomeMessage.Text = My.Resources.UserWelcome
End Sub
```

كما ترى فعملية الوصول إلى هذه المصادر عملية سهلة ويمكنك التقدم خطوة أخرى بتضمين أنواع مختلفة مثل الصور وأشياء أخرى وسيكون الوصول إليها بنفس السهولة كالنصوص فمثلا إذا قمت بإضافة ملف صورة LitwareLogo.png كمصدر وكذلك ملف xml مثلا Customers.xml لملف المصادر الخاص بالمشروع يمكنك الوصول إليهم بسهولة

```
Me.picLogo.Image = My.Resources.LitwareLogo

Dim xmlDoc As New Xml.XmlDocument
xmlDoc.LoadXml(My.Resources.Customers)
```

كما تلاحظ فإن فئات المصادر القوية strongly typed resource class تحول ملف الصورة إلى شئ صورة Image object يمكن تحميله مباشرة إلى صندوق الصورة PictureBox كما يحول ملف xml مضمن بسهولة إلى وثيقة xml

إعدادات الثقافة وتخصيص البرنامج محليا

من المتطلبات الشائعة لمشاريع البرامج أن يتم تخصيصها محليا وذلك بهدف تمكين أشخاص يتحدثون لغة معينة من التعامل معها فإذا كنت تقوم بكتابة برنامج وتريد توجيهه لمستخدمين يتحدثون الانجليزية والفرنسية في فيجول بايزيك2005 فإن الفريموورك لديها الكثير لتقدمه لك في هذا المجال فعندما تبدأ بكتابة تطبيقك المعتمد على الفريموورك ستحتاج لدعم التخصيص محليا localization ويجب عليك التعود على الفئة CultureInfo الموجودة ضمن نطاق الأسماء System.Globalization والتي تتبع اسم لغة تحدد اللغة المنطوقة فاسم اللغة الانكليزية En والفرنسية Fr كما أن الفئة CultureInfo تحمل معلومات أخرى حول المنطقة المستخدم فيها تلك اللغة فـ en-us من أجل US English و en-gb من أجل British English و fr-be من أجل Belgian French و إليك مثال ينشئ CultureInfo خاص بلغات مختلفة

```
Dim culture1 As CultureInfo = New CultureInfo("en-US")
Dim culture2 As CultureInfo = New CultureInfo("en-GB")
Dim culture3 As CultureInfo = New CultureInfo("fr")
Dim culture4 As CultureInfo = New CultureInfo("fr-BE")
```

وفي الحقيقة هناك شيئين object لـ CultureInfo يستدعيان انتباهك الأول CurrentCulture يمثل الثقافة الحالية بينما الثاني CurrentUICulture يمثل لغة الواجهة حيث يمكنك تحديد أسماء الثقافات للاثنين كالمثال

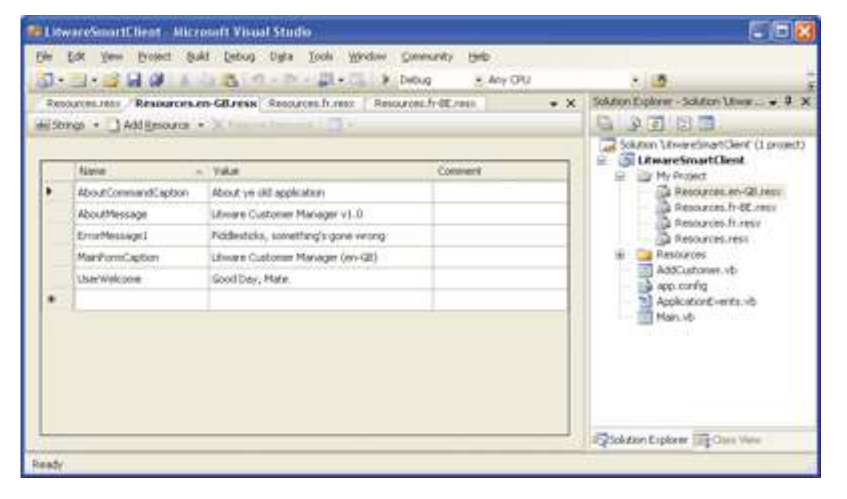
```
'*** determine current culture and current UI culture
Dim t As Thread = Thread.CurrentThread
Dim currentCulture As CultureInfo = t.CurrentCulture
Dim currentUICulture As CultureInfo = t.CurrentUICulture

'*** display cultures in console
Console.WriteLine("Current Culture: " & currentCulture.Name)
Console.WriteLine("Current UI Culture: " & currentUICulture.Name)
```

فالأول CurrentCulture لا يستخدم لجعل النصوص محلية ولكنه يحدد تنسيق التواريخ والأرقام والعملات فتغييره مثلا بين الانكليزية الأمريكية والبريطانية سيؤدي إلى نتائج غريبة مثلا عندما يتعلق الأمر بالعملات فالتغيير سيؤدي إلى نتائج خاطئة لهذا يتم تثبيت CurrentCulture حتى عندما نقوم بعمل برنامج بلغات مختلفة. وإذا كنت مصمما على تغيير CurrentCulture فتأكد من انك تستخدم أسماء ثقافات بأسماء مناطق محددة مثل en-us و en-gb و fr-be مثلا ومن ناحية أخرى فإنك ستحصل على أخطاء زمن التنفيذ عند استخدامك أسماء لغات مجردة مثل en أو fr لأنها لا تحتوي على تنسيقات التواريخ والأرقام ... الخ وستصبح متطلبات التنسيق غامضة.

والثاني CurrentUICulture هام للنقاش أكثر بخصوص جعل البرنامج محليا بسبب أن تغييره يؤثر على المسار الحالي current thread و كيفية تعامل الفريموورك مع مدير المصادر ResourceManager وتحميل المجمعات التي تحتوي على مصادر مضمنة كما أنه يؤثر في تحميل مجموعة المصادر الخاصة بلغة مستخدم معينة.

ولبدء عملية الـ localization يمكنك البدء بعمل عدة ملفات مصادر كل منها خاص بلغة معينة تريد دعمها فعلى سبيل المثال كنا نتحدث عن ملف مصادر على مستوى المشروع اسمه Resources.resx قم بنسخه وعمل نسخ عنه بواسطة النسخ واللصق داخل محرر المشروع Visual Studio Solution Explorer وبعد قيامك بعملية النسخ قم بتعديل الاسم بإضافة اسم اللغة قبل لاحقة الملف .resx مثلا Resources.fr-BE.resx من أجل الفرنسية في بلجيكا كالصورة



المجمعات البعيدة Satellite Assemblies

عندما تقوم بترجمة مشروع يتضمن مصادر مخصصة محليا لا يقوم فيجول ستوديو بترجمة جميع هذه المصادر إلى مجمع واحد وبدلا عن ذلك يقوم بترجمة هذه المصادر المخصصة محليا إلى مجمعات مستقلة كل منها يحتوي على تلك المصادر ولا يحتوي على كود وهذا ما يدعى بالمجمعات البعيدة Satellite Assemblies. و كل مجمع بعيد مرتبط مجمع رئيسي يدعى بالمجمع الطبيعي neutral assembly وهو يحتوي على كامل الكود والمصادر الطبيعية ويقوم بتحميل المجمع البعيد الضروري لتخصيص المصادر محليا حسب متطلبات المستخدم الحالي فمثلا فالبرنامج LitwareSmartClient.exe هو المجمع الطبيعي ويحتوي كامل كود التطبيق وهو مرتبط بعدة ملفات مصادر كل منها اسمه LitwareSmartClient.resources.dll وعندما تنشر المجمعات البعيدة مع المجمع الطبيعي في مجلدات ضمن مجلد التطبيق يتم نشرها وفق قواعد مجمعات الفريموورك وبشكل خاص فالمجمعات الخاصة بالثقافات المختلفة يتم نشرها في مجلدات بأسماء الثقافات المخصصة. فعلى سبيل المثال مجلد التطبيق LitwareSmartClient.exe يحتوي على مجلد باسم fr-BE خاص باللغة الفرنسية في بلجيكا وفيه ملف المصادر باسم LitwareSmartClient.resources.dll وطالما تم التقيد بهذه القواعد فإن محمل المجمع assembly loader بالتعاون مع فئة مدير المصادر ResourceManager class سيقوم بتحميل مجموعة المصادر المناسبة حسب الحاجة. ولحسن الحظ فإن فيجول ستوديو يعرف كيف يقوم بتسمية المصادر البعيدة ونشرها ضمن تركيب المجلدات الصحيح المتوقع من مجمل المجمع في الفريموورك وحتى تحصل على مستوى فهم أكثر لهذا التوزيع قم بترجمة compile مشروعك وتفحص بنية المجلدات الناتجة من مجلد التطبيق والمجلدات الفرعية فيه

تحميل المصادر المخصصة محليا

بعد قيامك بإنشاء وتحرير كافة ملفات المصادر المخصصة محليا وقيامك بترجمة compile مشروعك أصبح الوقت مناسبا للتحدث عن كيفية قيام برنامجك بتحميل مجموعة مصادر النصوص المخصصة محليا المفضلة من قبل المستخدم الحالي وإحدى الطرق للقيام بذلك هو الحصول على مرجع للمسار الحالي current thread وضبط CultureInfo جديد إلى الخاصية CurrentUICulture كالمثال

```
My.Application.ChangeUICulture("fr-BE")
```

في الكود المثال المرفق مع المقال تم إضافة إمكانية للمستخدمين لاختيار اللغة المفضلة لهم بالإضافة لمتابعة تغييرات اللغة عندما يغلق البرنامج ويعاد تشغيله فمع وجود إمكانية متابعة هذه الإعدادات بواسطة مفاتيح سجل النظام فإن فيجول ستوديو يوفر إمكانية للابتعاد عن سجل النظام باستخدام إعدادات التطبيق والتي يتم حفظها لكل مستخدم على حدى وفي المثال يقوم التطبيق بمتابعة هذه الإعدادات بالإعداد UserLanguagePreference في حدث البدء للتطبيق كما في المثال

```
'*** code in ApplicationEvents.vb
Namespace My
 Partial Friend Class MyApplication
   Private Sub MyApplication_Startup(ByVal sender As Object, _
       ByVal e As StartupEventArgs) Handles Me.Startup

     '*** initialize application by setting preferred language
     Dim lang As String = My.Settings.UserLanguagePreference
     My.Application.ChangeUICulture(lang)

   End Sub
 End Class
End Namespace
```

كما يوفر للمستخدم مجموعة من أزرار الانتقاء Radio buttons للتبديل بين اللغات المختلفة كما في المثال

```
'*** retrieve user's language preference
Dim lang As String = CType(sender, Control).Tag

'*** save user setting
My.Settings.UserLanguagePreference = lang
My.Settings.Save()

'*** change application's UI culture
My.Application.ChangeUICulture(My.Settings.UserLanguagePreference)

'*** call custom method to reload localized strings
LoadResources()
```

الذي يظهر جزءا من كود من معالج حدث يستجيب لطلب المستخدم بتغيير اللغة. فالآن وقد رأينا الكود الأساسي الذي يتيح للمستخدم تغيير اللغة فالفريموورك تستجيب لـ My.Application.ChangeUICulture بتحميل المجمع البعيد المناسب في المرة القادمة التي يقوم فيها مدير المصادر بقراءة نص من ملف المصادر على مستوى المشروع بعد استدعاء ChangeUICulture يمكن للتطبيق إعادة الاستعلام عن المصادر على مستوى التطبيق وتحميلهم على النموذج في نفس الكود الذي رأيناه سابقا ضمن الطريقة LoadResources

```
Me.Text = My.Resources.MainFormCaption
Me.lblWelcomeMessage.Text = My.Resources.UserWelcome
```

لاحظ أن محمل المجمع في الفريموورك سيحاول أولا إيجاد تطابق تام لاسم اللغة والمنطقة المطلوبة من اسم الثقافة في الكود واسم الثقافة في المجمعات البعيدة فإن لم يجد فسيحاول إيجاد شبيه من ضمن المجمعات المتوفرة فمثلا إذا كان المجمع يبحث عن fr-ca الفرنسية في كندا فإن

لم يجدها فسيبحث عن مجمع الفرنسية fr فإن لم يجده فسيعيد النص الموجود ضمن المجمع الطبيعي neutral assembly الذي يحتوي على مجموعة من المصادر بداخله وكما ترى فالفريموورك تعود للمجمع الطبيعي عند فشلها في تحديد المجمع الخاص بثقافة معينة .

عند قيامك بترجمة المجمع الطبيعي يمكنك تعليمه بخاصية attribute تحدد للفريموورك الثقافة الافتراضية فمثلا يمكنك تحديد خاصية في الملف AssemblyInfo.vb لإخبار التطبيق أنه عند استخدام الإعدادات الافتراضية فإنه يستخدم إعدادات اللغة الانكليزية مثلا

```
<Assembly: System.Resources.NeutralResourcesLanguage("en")>
```

تخصيص النموذج والتحكمات محليا

رأيت كيف يمكنك تخصيص مصادر النصوص على مستوى التطبيق وهذه التقنية تتضمن نسخ ومعالجة ملفات المصادر المخصصة محليا ويوفر لك فيجول ستوديو 2005 مساعدة إضافية عندما تريد تخصيص النصوص الخاصة بنموذج معين والتحكمات بداخله فكل نموذج Form له الخاصية Localizable والتي تأخذ قيمة True أو False فإن تم ضبطها إلى True فإن فيجول ستوديو سينشئ ويعالج مجموعة من ملفات المصادر المخصصة محليا من أجلك في الخلفية.

عندما تضبط الخاصية Localizable إلى True فاللغة الأساسية تكون default وعندما تضيف قيم للخصائص المختلفة ففيجول ستوديو يقوم بإضافتهم للمجمع الطبيعي وعندما تقوم بتغيير اللغة إلى لغة محددة مثل الفرنسية ببلجيكا سيقوم فيجول ستوديو بإنشاء ملف مصادر مخصص يتم ترجمته compiled ضمن المجمع البعيد تماما كما يتم بالنسبة لملفات المصادر على مستوى المشروع وبهذا يريحك فيجول ستوديو من التعامل مباشرة مع ملفات المصادر في صفحة الخصائص كخاصية Text للتحكم.

يحتاج فيجول ستوديو لإضافة كود إضافي للنموذج في الخلفية لدعم تخصيص النموذج محليا وبالتحديد فإنه يضيف كود لتحميل المصادر المخصصة محليا وضبطها من أجل النموذج والتحكمات وبدلا عن استخدام مدير المصادر ResourceManager يستخدم فيجول ستوديو فئة مشتقة منه تدعى ComponentResourceManager موجودة ضمن مجال الأسماء System.ComponentModel.

عندما يتم تحميل نموذج مخصص محليا فإن كل شئ متعلق بتحميل المصادر المخصصة محليا يتم عمله من أجلك من خلال الكود الذي يولده فيجول ستوديو تلقائيا وبالتحديد فإنه يوفر كودا يوفر تواجد Instance للفئة ComponentResourceManager التي تقوم بتحميل مجموعة المصادر المناسبة وتحديد القيم الخاصة بالتحكمات ومع ذلك فإن كان النموذج محملا وقام المستخدم بتغيير اللغة يجب عليك كتابة كود إضافي من أجل تجديد المصادر وفقا للإعدادات الجديدة كالمثال وذلك بإنشاء وتحديد تواجد للفئة ComponentResourceManager وتمرير نوع المعلومات له حول النموذج.

```
Dim crm As ComponentResourceManager
crm = New ComponentResourceManager(GetType(Main))
crm.ApplyResources(cmdAddCustomer, cmdAddCustomer.Name)
crm.ApplyResources(mnuFile, mnuFile.Name)
crm.ApplyResources(mnuFileAddCustomer, mnuFileAddCustomer.Name)
```

الفهرس